UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
A C A T L Á N

ANÁLISIS Y DISEÑO DE SISTEMAS ORIENTADOS A OBJETOS

T E S I N A

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA
LEANDRO BLANCAS AGUILAR

ASESOR
LIC. JUAN CARLOS RENDÓN AGUILAR

SANTA CRUZ ACATLÁN, ESTADO DE MÉXICO.

AGRADECIMIENTOS

A Dios.

Por todas las bendiciones que he recibido de él a lo largo de la vida que me ha regalado.

A mi asesor: Lic. Juan Carlos Rendón Aguilar

Por su comprensión y tiempo dedicado a la revisión de este trabajo, por sus valiosos comentarios y orientación brindada.

A la máxima casa de estudios. U.N.A.M.

Por brindarme la oportunidad de formar parte de la gran institución que es.

DEDICATORIA

A mi linda esposa Verónica.

Por el inmenso amor que siempre me has demostrado con tus acciones, por la enorme confianza que depositaste en mi, por el inmenso apoyo que me has brindado en los momentos más difíciles, porque no importando la hora siempre me diste palabras de aliento para seguir adelante, pero más que nada por tu amor y paciencia. Gracias.

Te Amo.

A mis dos princesitas: Diana Cristina y Andrea Montserrat.

Porque con su ternura e inocencia siempre fuerón y serán motivo de todo mi esfuerzo, gracias por su alegría porque con sus caritas sonrientes me inyectaron de vitalidad para lograr lo inalcanzable y porque siempre que estaba trabajando frente al computador me proveían constantemente de dulces, papitas, coca-cola y mucho amor.

Las adoro inmensamente princesitas.

A mis padres: Eva y Leandro

Les agradezco sinceramente todo el apoyo incondicional que he recibido de ustedes, por la confianza y el amor que ahora me permiten realizar este sueño que también es suyo y por enseñarme los valores de la familia, que son la herencia más importante que me pueden dar y porque han hecho de mi un hombre de bien.

Leandro Blancas Aguilar.

ÍNDICE DE CONTENIDO

CAPÍTULO II CONCEPTOS FUNDAMENTALES DEL MODELO DE ORIENTACIÓN A OBJETOS PARA EL DESARROLLO DE SISTEMAS

CAPÍTULO III PROCESO DE ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

PRÓLOGO

El presente trabajo de investigación por tener el carácter de Tesina y servir como requisito para obtener el título de Lic. en Matemáticas Aplicadas y Computación, no es un trabajo exhaustivo ni pretende ser un tratado sobre el análisis y diseño de sistemas, sin embargo cumple con los objetivos de proveer el entendimiento de los conceptos fundamentales del análisis y diseño de sistemas orientados a objetos, de tal manera que facilite la comprensión de un problema y permita documentar la arquitectura de un sistema en desarrollo. Constituyendo así, un apoyo durante el ciclo de vida del desarrollo de un sistema orientado a objetos.

Este trabajo también pretende servir como fuente de información para los estudiantes que se inician en el arte del desarrollo de sistemas de software.

INTRODUCCIÓN

La "orientación a objetos" será la más importante de las tecnologías que surjan en los años noventa.
Bill Gates, Presidente de Microsoft Corp.

Nadie puede negar que vivimos en un mundo de sistemas, los cuales están dentro de otros sistemas que son parte de otros aún mayores y por lo tanto, toda nuestra actividad cotidiana personal y profesional tiene algún impacto sobre los diversos sistemas de los cuales formamos parte.

Es por esta razón que este trabajo comienza con introducir al lector al estudio de los sistemas desde el punto de vista de la teoría general de sistemas y de la naturaleza de los mismos, para posteriormente particularizar en los sistemas de información, los cuales nos interesan como profesionales de las ciencias de la computación.

El presente trabajo se enfoca en el análisis y diseño orientado a objetos más que en la programación, debido a que la historia del desarrollo de sistemas de software nos ha mostrado que así es la tendencia, ya que en la década de los años 70 se consideraba como un problema de programación, incluso durante esta década se creo el lenguaje SMALLTALK, uno de los lenguajes orientado a objetos más puro.

Al principio de la década de los años 80 se puso de manifiesto el interés por las interfases gráficas de usuario (Graphic User Interface), los pioneros comerciales más conocidos fueron Xerox y, después, Apple, introdujeron las interfases universales WIMP[1], la influencia de este estilo se extendió a las máquinas IBM PC y PS/2 que fueron equipadas con Microsoft Windows.

---

[1] **WIMP** viene de Windows, Icons, Mice and Pointers (ventanas, iconos, ratones y punteros) y se refiere al estilo de interfase gráfica de usuario que los utiliza.

La década de los años 90 se caracteriza porque ha permitido la evolución de los métodos orientados a objetos, esta evolución se basa en uno de los principales objetivos de la Ingeniería de software; la reutilización de componentes de software y el mejoramiento de la calidad del mismo, esto aunado a la creciente complejidad del software obligó a poner más atención en los aspectos de análisis y diseño dentro del ciclo vital del desarrollo de sistemas, es decir, se puso más interés en la especificación de los requerimientos, ya que la correcta especificación de éstos permite reducir los errores tanto en la especificación como en la interpretación y en la implementación.

Al respecto Barry Boehm[2], considera que toma 100 veces más esfuerzo corregir errores en especificaciones y requerimientos cuando se ha llegado a la etapa de codificación que el esfuerzo que toma corregir esos mismos errores si son descubiertos cuando se están produciendo esas especificaciones o requerimientos.

En el capítulo II, se presentan los conceptos básicos asociados con la terminología empleada en la orientación a objetos, también se trata de introducir el modelo de orientación a objetos como medio para manejar la complejidad inherente del software y se hace énfasis en la importancia de identificar correctamente las clases y objetos de una aplicación ya que esta es una tarea clave en el análisis orientado a objetos. Se presentan diferentes enfoques de clasificación de objetos, que sirven como el fundamento teórico para el análisis orientado a objetos y se presentan a la vez diferentes enfoques de éste, que proporcionan útiles técnicas para la identificación de clases y objetos.

En el capítulo III, se explicará la importancia de contar con una notación estándar y bien definida. En particular se introducirá la notación de Grady Booch así como la necesidad de contar con múltiples vistas de un sistema para poder tener una mejor comprensión del sistema que se pretenda desarrollar.

Se verá que la notación de Booch comprende un conjunto de modelos, los cuales representan cada uno una visión diferente de un sistema orientado a objetos y están representados mediante diferentes diagramas que sirven para describir la existencia y el significado de las clases y objetos así como las interrelaciones entre ellos y para definir la arquitectura de un sistema.

---

[2] Citado por Cárdenas García, Sergio R. de los laboratorios Bell, AT&T, E.E.U.U. en el artículo: *Una panorámica de la ingeniería de software* en la revista Soluciones Avanzadas, núm. 8, marzo-abril, México, 1994. pp. 4-6.

También se explicará el proceso de desarrollo incremental e iterativo considerando el propósito, los productos y las actividades de cada fase. Se verá que el proceso es iterativo en el sentido que implica el mejoramiento sucesivo de una arquitectura orientada a objetos, y es incremental en el sentido de que cada pasada por el ciclo análisis/diseño/evolución conduce a mejorar las decisiones estratégicas y tácticas tomadas.

En el capítulo IV, se abordará un ejemplo real de una aplicación actual del área agropecuaria, con el fin de aplicar la notación y ejemplificar el proceso de desarrollo descrito en el capítulo III. Se escogió éste tipo de aplicación debido a mi trabajo en el Colegio de Postgraduados en Ciencias Agrícolas, donde existen varios ejemplos de sistemas cuya complejidad demanda de las técnicas de análisis y diseño orientado a objetos.

Además este tipo de aplicaciones demuestra que nuestro trabajo puede ser multidisciplinario, esto significa que el trabajo como analista de sistemas implica tener más que el conocimiento de las técnicas para desarrollar sistemas de software y la habilidad para dibujar diagramas, demanda de nosotros habilidad para entender los requerimientos de expertos en otras áreas diferentes a la nuestra, también se requiere facilidad para el manejo de personas para poder entrevistar a los usuarios y obtener de ellos sus verdaderas necesidades de información, así como tener capacidad de mediador ya que muchas veces uno se encuentra en medio de usuarios, administradores, programadores y otros diversos participantes los cuales muchas veces están en desacuerdo entre sí y el analista debe ser capaz de exponer sus propias ideas sobre cómo debe ser el sistema o cuales funciones debe cumplir.

Finalmente se tratan algunas tendencias en el software orientado a objetos, principalmente están dirigidas hacia la reutilización de código depurado y comprobado contenido en bibliotecas de clases. Se habla de la utilización de las técnicas orientadas a objetos en aplicaciones de inteligencia artificial y de sistemas expertos.

CAPÍTULO I

INTRODUCCIÓN AL ANÁLISIS Y DISEÑO DE SISTEMAS

1.1 Objetivos

Los objetivos de éste capítulo son:

Introducir al lector al estudio de los sistemas y presentar un marco de referencia para su análisis y diseño.

Abordar el tema de la complejidad y el porque es una característica de los grandes sistemas de software.

Introducir el método de orientación a objetos como medio para manejar dicha complejidad.

1.2 Marco teórico conceptual

Para introducir al lector al estudio de los sistemas se hará uso de la *Teoría General de Sistemas*. y del *Enfoque de Sistemas*, el cual es la aplicación de la teoría general de sistemas para la solución de problemas del mundo real. De hecho el enfoque sistémico o enfoque de sistemas también recibe el nombre de *Teoría General de Sistemas Aplicada.*[1]

Existen varios principios de esta teoría que pueden ser aplicados a los sistemas computarizados. Es en este contexto, que el concepto de sistema no se tratará en un principio como lo referente al procesamiento de datos o de sistemas de información ni de la manera en que estos se desarrollan, sino que se tratará desde un punto de vista más general, para posteriormente particularizar en los métodos y técnicas para desarrollar sistemas, pero ahora si, desde un punto de vista informático.

1.2.1 Desarrollo Histórico

La fuente de la teoría general de sistemas puede remontarse probablemente, a los orígenes de la ciencia y la filosofía. Esto lo podemos apreciar en el pensamiento de algunos filósofos como el caso del alemán George Wilhelm Friederich Hegel (1770-1831), a quien se le atribuyen las siguientes ideas.

1) El todo es más que la suma de las partes.

2) El todo determina la naturaleza de las partes.

3) Las partes no pueden comprenderse si se consideran en forma aislada del todo.

4) Las partes están dinámicamente interrelacionadas o son independientes

Para el propósito del presente trabajo, será suficiente situarse en el año de 1954, cuando se organizó la Society for the Advancement of the General Systems Theory (Sociedad para el avance de la teoría general de sistemas). En 1957, se cambió el nombre de la sociedad por su nombre actual, la Society for General Systems Research (Sociedad para la investigación general de sistemas). En 1956 la sociedad publicó el libro, *sistemas generales*, en el cual *Ludwig von Bertalanffy*[2]

---

[1] Van Gigch, John P. *Teoría General de Sistemas*, Trillas, México, 1987. pp. 45.

[2] Se considera el padre de la teoría general de sistemas. En la obra *Sistemas Generales*, también predijo que el concepto de sistema se convertiría en el punto de apoyo del pensamiento científico moderno.

escribió un artículo en el que presentó los propósitos de esta disciplina de la siguiente manera:

"a) Existe una tendencia general hacia la integración de las ciencias naturales y sociales.

b) Tal integración parece centrarse en una *teoría general de sistemas.*

c) Tal teoría puede ser un medio importante para llegar a la teoría exacta de los campos no físicos de la ciencia.

d) Desarrollando principios unificados que van "verticalmente" a través de los universos de las ciencias individuales, esta teoría nos acerca al objetivo de la unidad de la ciencia (la unificación del fragmentado conocimiento humano sobre el mundo, a través de un enfoque unitario).

e) Esto puede conducir a la integración muy necesaria de la educación científica."[3]

1.2.2 Otras fuentes

La teoría general de sistemas es el resultado de otras contribuciones fundamentales, como son las siguientes[4]:

1) John von Neumann (1948), quien desarrolló una teoría general del autómata y delineó los fundamentos de la *inteligencia artificial*

2) El trabajo de C.E. Shannon, *Teoría de la información* (1948), en el cual se desarrolló el concepto de cantidad de información alrededor de la *Teoría de las comunicaciones.*

3) Norbert Wiener (1948), en *Cibernética*, relacionó entre sí los conceptos de entropía, desorden, cantidad de información e incertidumbre, y acentuó su importancia en el contexto de los sistemas.

4) Ross W. Ashby (1956), desarrolló los conceptos de cibernética, autorregulación y autodirección, basándose de las ideas originales de Wiener y Shannon.

---

[3] Van Gigch, John P. *Teoría General de Sistemas*, Ed. Trillas, México, 1987, p. 65.

[4] *Ibidem.* p. 68.

Todas estas ideas derivadas de la teoría general de sistemas nos ayudan a comprender su naturaleza, esto es importante porque si comprendemos algo acerca de la teoría general de sistemas entenderemos mejor los sistemas de información computarizados.

De esta manera, empecemos con la definición del término básico: **sistema**

1.3 El concepto de sistema

Diferentes autores han definido el concepto sistema, y seguramente todas las definiciones sean correctas. Empecemos citando a John P. Van Gigch, quien al respecto dice:

> "Un sistema es una reunión o conjunto de elementos relacionados.
>
> Los elementos de un sistema pueden ser **conceptos**, en cuyo caso estamos hablando de un sistema conceptual. Un lenguaje es un ejemplo de un sistema conceptual. Los elementos de un sistema pueden ser **objetos**, como por ejemplo, una máquina y los componentes que la integran. Los elementos de un sistema pueden ser **sujetos**, como los integrantes de un equipo de fútbol. Finalmente un sistema puede integrarse de conceptos, objetos y/o sujetos. Por lo tanto un sistema es un agregado de entidades, vivientes o no vivientes, o ambas que a su vez pueden formar **subsistemas**.
>
> Uno de los problemas al tratar con los sistemas se deriva de nuestra incapacidad para saber que tanto podemos 'descomponer' un sistema en subsistemas componentes, o que tanto podemos 'componer' u 'organizar' un sistema en sistemas mas grandes."[5]

Ahora bien, también existen las siguientes definiciones de sistemas:

> "Sistema: un agrupamiento de componentes cuyo comportamiento conjunto depende tanto del de las partes como de la forma en que éstas interactuan entre sí."[6]

El **ANSI** (American National Standards Institute) lo define de la siguiente manera:

> Un conjunto de métodos, procedimientos o técnicas unidas por una interacción regulada para formar un todo organizado.

---

5 *Ibidem.* pp. 16-17.

6 Gerez, Victor *et. al. Introducción al análisis de sistemas e investigación de operaciones*, Ed. Representaciones y Servicios de Ingeniería, México, 1978. p. 6.

La International Organization for Standarization Technical Committee (El Comite Técnico de la Organización Internacional para la Estandarización) también lo define como:

> "Un conjunto organizado de hombres, máquinas y métodos que se requieren para lograr un conjunto de funciones específicas."[7]

Si seguimos buscando, encontraremos muchas más definiciones sobre el tema, sin embargo, podemos ver que las que se han citado hasta este punto, tienen algunos elementos en común y podríamos establecer nuestro propio entendimiento del concepto sistema, sin que este difiera de lo que en esencia es.

Entonces, podemos partir de que un sistema es un conjunto de elementos, ya sean vivientes (personas, células, instituciones, empresas, etc.) y/o no vivientes (objetos, métodos, conceptos, técnicas, máquinas, dinero, etc.), que interactuan entre sí para lograr uno o varios objetivos comunes.

1.3.1 Tipos de sistemas

Existen muchos tipos diferentes de sistemas; de hecho casi todo con lo que tenemos contacto cotidianamente es un sistema, o bien forma parte de uno de ellos. Para entender la naturaleza de los sistemas computacionales, empecemos por dividir todos los sistemas en dos categorías: ***sistemas naturales*** y ***sistemas hechos por el hombre.***[8]

Los ***sistemas naturales*** se subdividen en dos categorías básicas: *sistemas físicos* y *sistemas vivientes*. En los *sistemas físicos* se incluyen por ejemplo:

- Sistemas estelares: galaxias, sistemas solares, etc.
- Sistemas geológicos: ríos, cordilleras, etc.

En los *sistemas vivientes* se incluyen toda la variedad de plantas y animales que constituyen nuestro entorno, al igual que la raza humana. Esta categoría también comprende organismos vivientes individuales, por ejemplo: tribus, grupos sociales, compañías, naciones.

---

7 Squire, Enid. *Introducción al diseño de sistemas*, Ed. Fondo Educativo Interamericano. México, 1984. p.1.

8 Yourdon, Edward. *Análisis Estructurado Moderno*, Ed. Prentice Hall Hispanoamericana, México, 1993. pp. 13.

En los ***sistemas hechos por el hombre*** se incluyen:

- Sistemas sociales: organizaciones de leyes, doctrinas, costumbres, etc.
- Sistemas de transporte: redes de carreteras, de trenes, aerolíneas, etc.
- Sistemas de comunicaciones: teléfono, télex, etc.
- Sistemas de manufactura: fábricas, líneas de ensamble, etc.
- Sistemas financieros: contabilidad, inventarios, bolsa de valores, etc.
- Sistemas de información basados en computadoras.

La mayoría de estos sistemas incluyen a las computadoras como parte de ellos. Es importante señalar que muchos de estos sistemas existían desde antes que surgieran las computadoras, también algunos de ellos continúan sin computarizarse y tal vez sigan así por mucho tiempo.

Muchas veces, nosotros como analistas de sistemas, suponemos que todo sistema con el que nos topemos debe computarizarse, pero el verdadero papel del analista es estudiar un sistema para determinar su esencia, es decir, el comportamiento que este requiere, independientemente de la tecnología que se piense utilizar para implementarlo.

A continuación se analizarán las características de los sistemas en general, con el fin de entender la importancia que tiene este concepto y el impacto que tiene en nuestra vida cotidiana.

1.3.2 Características

Como características distintivas de los sistemas que interesan a los analistas de sistemas, se pueden citar las siguientes[9]:

1) La mayoría de los sistemas han sido implementados por el hombre e incorporan equipos y procedimientos (por ejemplo, el sistema de transporte colectivo). Sin embargo, no se puede excluir del enfoque sistémico a aquellos que no tienen estas características, como por ejemplo: un sistema ecológico, o la cuenca de un río.

2) Estos sistemas son complejos y están formados de muchas partes que interactuan entre sí; el comportamiento de cada componente y la estructura de su interconexión afectan el comportamiento del sistema en su conjunto.

---

[9] Gerez, Víctor *et. al. Introducción al análisis de sistemas e investigación de operaciones*, Ed. Representaciones y Servicios de Ingeniería, México, 1978. p. 7-9.

3) Estos sistemas son frecuentemente semiautomáticos, es decir, algunas de sus funciones las realizan operadores humanos, y otras son realizadas por máquinas y/o computadoras. Las variables que afectan a estos sistemas son por lo general probabilísticas. Esta última característica impide predecir con exactitud el comportamiento del sistema, por lo que dicha predicción puede hacerse solamente en términos probabilísticos.

4) Otra característica de los sistemas que requiere la sociedad, es la escala; son grandes desde cualquier punto de vista. Los niveles de inversión que requieren para su implementación pueden alcanzar una parte importante del presupuesto de un país (Por ejemplo, el sistema educativo nacional o el sistema nacional de salud).

5) La escala de tiempo de estos proyectos puede llegar a ser muy grande, de tal manera que el intervalo de tiempo que transcurre entre la inversión y el inicio de la generación de beneficios del proyecto, también es muy grande.

6) Estos grandes sistemas tienen además implicaciones económicas que no se presentan en proyectos de menor tamaño. El objetivo de estos sistemas es frecuentemente proveer un satisfactor social, y no un bien de consumo. Los bienes sociales requieren de decisiones compartidas por todos los miembros de la sociedad.

   Los niveles de inversión que se requieren para la realización de estos proyectos de beneficio social son muy fuertes, por lo que los gobiernos recurren a la estrategia de restringir el consumo y propiciar el ahorro. Esta situación produce frecuentemente tensiones sociales, sobre todo en el lapso de tiempo que transcurre entre el inicio del proyecto y la percepción de los beneficios del mismo. Por lo tanto las decisiones en este tipo de proyectos deben tomarse con especial cuidado y ser de carácter participativo por los sectores de la sociedad a los cuales involucra o beneficia. Para no causar intranquilidad social es necesario informar ampliamente a la población sobre cuales son las alternativas del proyecto, sus costos y beneficios. (Pensemos en el proyecto de la construcción de un aeropuerto y los beneficios económicos que este puede traer para la región donde se ubicaría, o en la construcción de una presa, o de una planta termoeléctrica).

   Muchos aspectos de estos grandes sistemas tienen que ser planeados, implementados y operados por el sector público, ya que los niveles de inversión superan a la capacidad financiera del sector privado. Esto representa una mayor participación del sector público en la economía del país. Sin embargo, hay que tener en cuenta que los beneficios sociales que esta participación representa se verán seriamente afectados si no se planean, implementan y operan de una manera eficiente.

Otra característica de los grandes sistemas, derivada de su complejidad y tamaño, es el impacto ecológico, que se debe tener cada vez más en cuenta para la planeación. En algunos casos en que no se toma en cuenta el enfoque sistémico para la planeación, conlleva a implementaciones cuyo impacto ecológico se puede corregir con relativa facilidad, pero también existe el riesgo de que no suceda así, lo cual significaría crear problemas posiblemente más graves a la sociedad. A medida que aumenta el tamaño de los sistemas planeados, los efectos negativos pueden ser irreversibles, y llegar inclusive a anular el beneficio socio-económico producido por el proyecto.

A continuación se resumen las características de los sistemas, descritas anteriormente:

- La mayoría son implementados por el hombre.
- Son complejos y sus partes interaccionan.
- Frecuentemente son semiautomáticos.
- Son de gran escala.
- El tiempo de desarrollo es también grande.
- Con frecuencia producen satisfactores públicos.
- Los grandes proyectos requieren de fuertes inversiones, por lo que algunos deben ser implementados por el sector público.
- tienen impacto sobre la naturaleza.

1.3.3 Sistemas bien y mal planeados

La historia del desarrollo tecnológico de la humanidad esta llena de implementaciones de sistemas cuyos efectos positivos han sido contrarrestados por los efectos negativos del mismo. Un ejemplo tomado de la antigüedad romana, es el sistema de agua potable de Roma. El plomo de sus tuberías, se afirmó, produjo un envenenamiento lento de la población y cambios genéticos, a lo que algunos historiadores atribuyen la decadencia del imperio.

Considerando épocas más recientes, dos ejemplos muestran claramente la diferencia de la evolución de dos sistemas: uno planeado desde un principio tomando en cuenta su finalidad; el sistema de telefonía, y el otro que creció en forma no planeada; el de transportación individual por automóvil.

El sistema telefónico, formado por una red integrada de componentes y equipos, ha hecho posible la comunicación inmediata entre dos puntos del sistema. Este sistema trabaja de manera eficiente y cumple con su finalidad, ya que desde un principio fue concebido como un sistema y no como un conglomerado de componentes independientes.

Supongamos por un momento que el sistema telefónico hubiera evolucionado de una manera no planeada, sin tomar en cuenta los requerimientos sistemáticos. Algún industrial hubiese empezado a hacer teléfonos a pequeña escala y los hubiese podido vender a sus clientes. Estos hubiesen contratado a algún electricista para conectar entre sí los teléfonos de las personas con las que en esos momentos deseaban comunicarse, el electricista hubiera colgado los cables de los postes públicos, uno por cada persona con la que deseaba establecer comunicación. En la medida que los usuarios se hubieran acostumbrado a hablar con amigos y otras personas, se hubieran conectado más y más cables de teléfonos. Tarde o temprano algún empresario hubiera pensado en establecer una central, permitiendo a cada abonado conectarse a ella por una tarifa, y él de alguna forma se hubiera encargado de establecer la conexión. Con el tiempo estas centrales se multiplicarían compitiendo entre sí pero sin poder comunicar a los abonados de una central con los de otra diferente o cobrando una tarifa extra por hacerlo. Con este tipo de sistemas tal vez podríamos hacer llamadas de larga distancia ocasionalmente y con una calidad pésima y lo que es peor con un costo muy elevado. Los requerimientos de la sociedad de contar con un sistema rápido, confiable y económico de comunicación, tarde o temprano habrían obligado a adoptar el enfoque sistémico en la planeación de este servicio.

Afortunadamente, el servicio telefónico se desarrolló desde un principio tomando en cuenta el enfoque sistémico.

Este ejemplo puede parecer cómico, pero no difiere mucho del sistema de transportación en automóvil, siendo éste causa de congestionamientos en las grandes ciudades y en gran medida de la contaminación de la atmósfera, problema que cotidianamente vivimos en la ciudad de México y sufren en otras ciudades como Río de Janeiro, Santiago de Chile, Tokio, Londres, inclusive algunas ciudades de los Estados Unidos, país donde se concibió el modelo más popular de automóvil, creado por Henry Ford.

Se escogieron estos ejemplos porque existe entre ellos un gran contraste. El sistema de transportación individual está integrado por automóviles, carreteras, calles, refaccionarias, gasolineras, señales de tránsito, etc., cabe señalar que algunos de estos elementos pueden ser subsistemas a la vez. Todo este conglomerado de subsistemas no ha sido debidamente planeado como un sistema integral, tomando en cuenta las interrelaciones que existen entre ellos y el objetivo del mismo: transportar personas eficientemente. Este sistema creció al azar sin planeación sistemática, sin ninguna directriz.

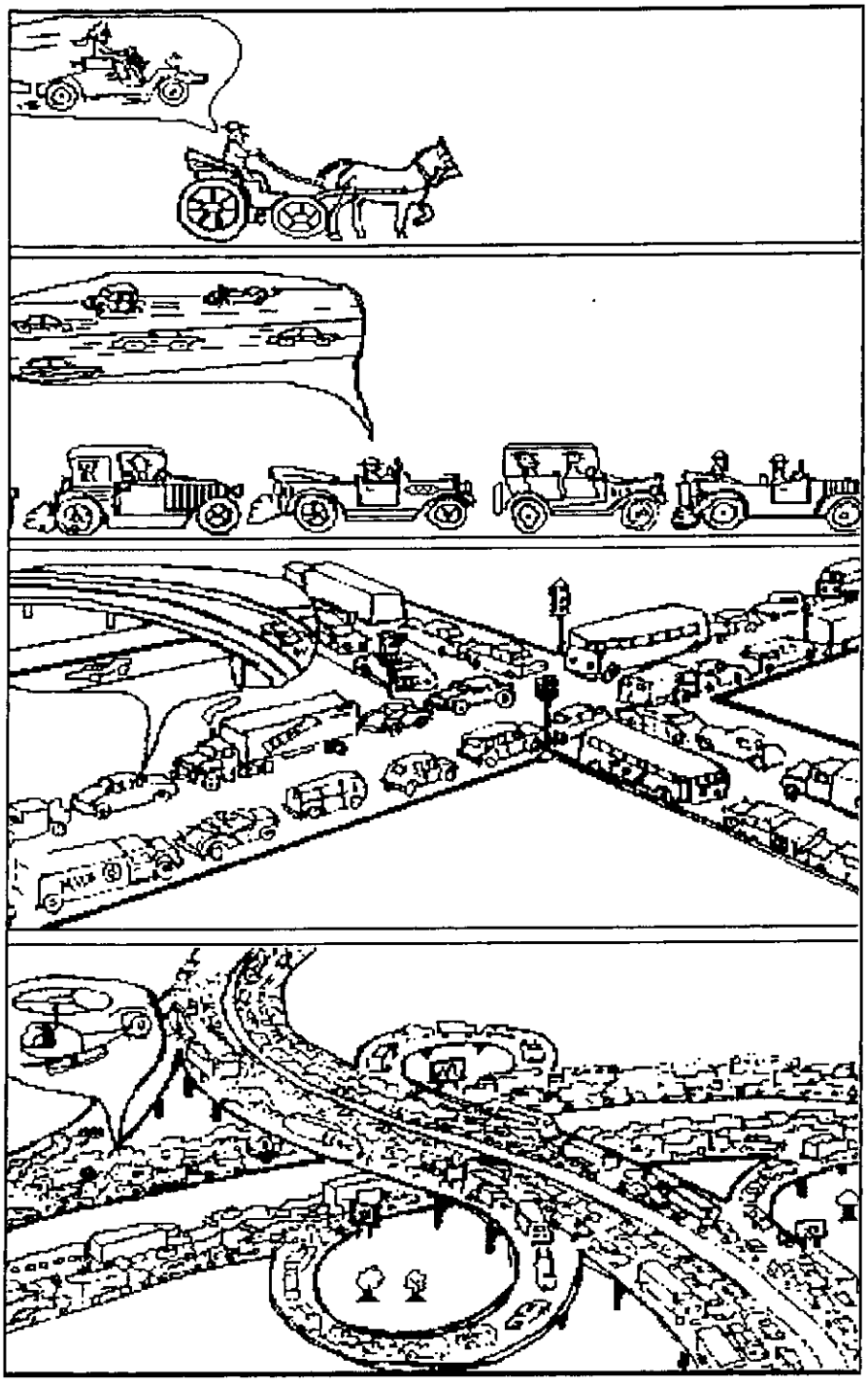
Ambos sistemas cumplen con su objetivo, el de telefonía con el de comunicar y transmitir voz, en un principio, después, fue evolucionando hasta que en la actualidad transmite no sólo voz sino también datos e imágenes y se ha integrado a redes mundiales de comunicación siendo una parte fundamental, como por ejemplo en Internet. También se han creado nuevos subsistemas como el de telefonía celular. Y puede seguir creciendo, no se pueden asegurar sus límites.

Por su parte, el sistema de transportación individual en automóvil al principio si cumplía con sus objetivos; transportar a personas y recorrer más distancia en menos tiempo. Pero estos objetivos han sido rebasados y ahora es muchas veces contraproducente transportarse individualmente en automóvil y hay ocasiones en que sucede lo contrario, es decir, se tarda uno más tiempo en recorrer una determinada distancia. Existen automóviles que han sido diseñados para correr a velocidades superiores a los 200 Km/hr, sin embargo se encuentran rodando lentamente, y hasta parados. El sistema causa miles de accidentes, ha hecho perder miles de horas-hombre al día con sus consecuentes repercusiones económicas, contamina el aire que respiramos, e incluso es responsable directo de daños en nuestra salud. Como consecuencia ha sido necesario crear otros sistemas para aliviar los problemas creados por éste, por ejemplo, en nuestro país se creó el *Sistema de Transporte Colectivo* METRO y los programas de restricción de circulación de vehículos una o dos veces por semana (programa de hoy no circula), programas de verificación vehicular, etc.

En la figura 1.1, tomada de Gerez, Víctor[10], se ilustra el tiempo cada vez mayor que tarda una persona en ir al trabajo, como consecuencia de un sistema de transportación individual no planeado en forma integral.

Esta comparación entre la evolución de dos sistemas públicos ilustra adecuadamente la importancia del enfoque sistémico en la planeación, implementación y operación de un sistema.

[10] Gerez, Víctor *et. al. Introducción al análisis de sistemas e investigación de operaciones*, Ed. Representaciones y Servicios de Ingeniería, México, 1978. p. 13.



Velocidad promedio del tráfico: 10 Km./hr.

Tiempo para llegar a casa: (3 Km.): 20 min.

Velocidad promedio del tráfico: 19 Km./hr.

Tiempo para llegar a casa: (8 Km.): 25 min.

Velocidad promedio del tráfico: 22 Km./hr.

Tiempo para llegar a casa: (13 Km.): 35 min.

Velocidad promedio del tráfico: 24 Km./hr.

Tiempo para llegar a casa: (19 Km.): 48 min.

Figura 1.1 Al mejorar las carreteras y los vehículos, la gente tarda más tiempo en trasladarse, y no hay una mejoría sensible en el tráfico y consecuentemente se crearon problemas como la contaminación ambiental. No se planeó en forma eficiente el sistema de transportación individual por automóvil.

1.3.4 Clasificación

Un modo conceptual de clasificar los sistemas se basa en dos criterios distintos, que son los siguientes:

1.- Nivel de predictibilidad. este criterio está basado en un doble esquema: determinista y probabilista.

2.- Nivel de complejidad. Adoptando este criterio, es posible clasificar los sistemas en tres categorías: sencillos, complejos y sumamente complejos.

Cuando es posible pronosticar con certeza la manera en que responderá un sistema o como funcionará en circunstancias diferentes, decimos que el sistema es **determinista**. Es decir, no hay ninguna duda sobre los resultados. Debido a la cantidad de información disponible y al último estado del sistema, será posible predecir sin riesgo de equivocarse, el estado subsecuente del sistema. A la inversa, cuando no es posible pronosticar con certeza los resultados de un sistema en particular, dicho sistema es **probabilista**.

Un **sistema determinista sencillo** es aquel que contiene pocos subsistemas e interrelaciones y presenta un comportamiento completamente predecible. Por ejemplo, consideremos unos balines de acero colocados sobre una pista, podría considerarse un sistema determinista sencillo. Sin embargo, una vez que los balines salen de la pista, sus caminos se vuelven probabilistas. Una hilera de máquinas en la línea de producción se puede estudiar y distribuir de manera que se reduzca al mínimo la distancia que recorren los materiales, cuando se hace necesario estudiar lo que ocurre realmente cuando los materiales comienzan a fluir, el sistema se torna inmediatamente probabilista.

Las mismas consideraciones son aplicables cuando estudiamos un **sistema determinista complejo.** La computadora es un sistema complejo; pero es determinista en el sentido de que hará aquello para lo cual fue programada.

Un **sistema sencillo** también puede ser **probabilista**, por ejemplo, lanzar una moneda al aire es un sistema sencillo pero es a la vez notoriamente probabilista. Un sistema de control de calidad, que pronostica el número de piezas defectuosas es un sistema sencillo, pero a la vez probabilista.

Una empresa es un **sistema probabilista complejo** cuyo objetivo es la obtención de utilidades. En las actividades de la empresa se toman decisiones que influyen en varios de los subsistemas que componen el sistema, y la influencia de tales decisiones, modifica hasta cierto punto las operaciones de la empresa con el fin de alcanzar cierto nivel de utilidades, todo lo cual es probabilista.

Un sistema tan complicado, que resulte casi indescriptible entra en la categoría **sumamente complejo.** Debido a que esta quinta categoría es tan compleja, en consecuencia no podrá ser determinista. Por lo tanto, no existe la clasificación **sistema determinista sumamente complejo.**

Sin embargo, en la categoría **probabilista sumamente complejo**, los resultados son diferentes por completo. Por ejemplo, la economía nacional es tan compleja y tan probabilista, que es muy difícil que se llegue a describir completamente.

En la actualidad muchas de la organizaciones se han vuelto muy complejas, hasta el punto de que muchas de ellas entran en la categoría de probabilistas sumamente complejas. De aquí surge la necesidad de comprender y mejorar las metodologías de análisis y diseño de sistemas para afrontar esa complejidad. El cuadro 1.1, resume la clasificación de sistemas descrita hasta este punto.

Sistema	Sencillo	Complejo	Extremadamente Complejo
DETERMINISTICO	Balines de acero en una pista	Computadora	No hay clasificación
	Línea de ensamble de producción	Bodega automatizada	
PROBABILISTICO	Lanzar una moneda al aire	Pequeña empresa	Gran empresa
	Sistema de control de calidad	Sistema de inventario	Economía nacional

Cuadro 1.1 Clasificación de los sistemas según su nivel de predictibilidad y complejidad.

Hasta este momento se han tratado los sistemas desde el punto de vista de la teoría general de sistemas, sin embargo, nuestro interés debe enfocarse hacia los sistemas de información, su análisis y diseño; así que ahora estudiaremos el significado de la información y los elementos que los constituyen así como el ámbito en que estos se desarrollan.

1.4 ¿Qué es la información?

La información esta compuesta por datos, colocados en un contexto significativo, útil y dirigidos hacia un receptor el cual los utilizará para tomar decisiones. Los datos pueden ser números, imágenes, texto, voz, documentos, siempre organizados en un contexto significativo; muchas veces el término datos se emplea para abarcar a todos estos componentes de la información.

1.4.1 Ciclo de la información

En el ciclo de la información los datos se procesan mediante modelos para crear nueva información; el receptor la recibe y luego toma una decisión y actúa; esto genera resultados que se interpretan como acciones o eventos, que a su vez crean diversos datos que se capturan y sirven nuevamente de entrada; y el ciclo se vuelve a repetir. En la figura 1.2, se presenta el ciclo de la información, cabe destacar que los datos que se van a procesar pueden ser datos de entrada, estar almacenados en una base de datos o ambos.

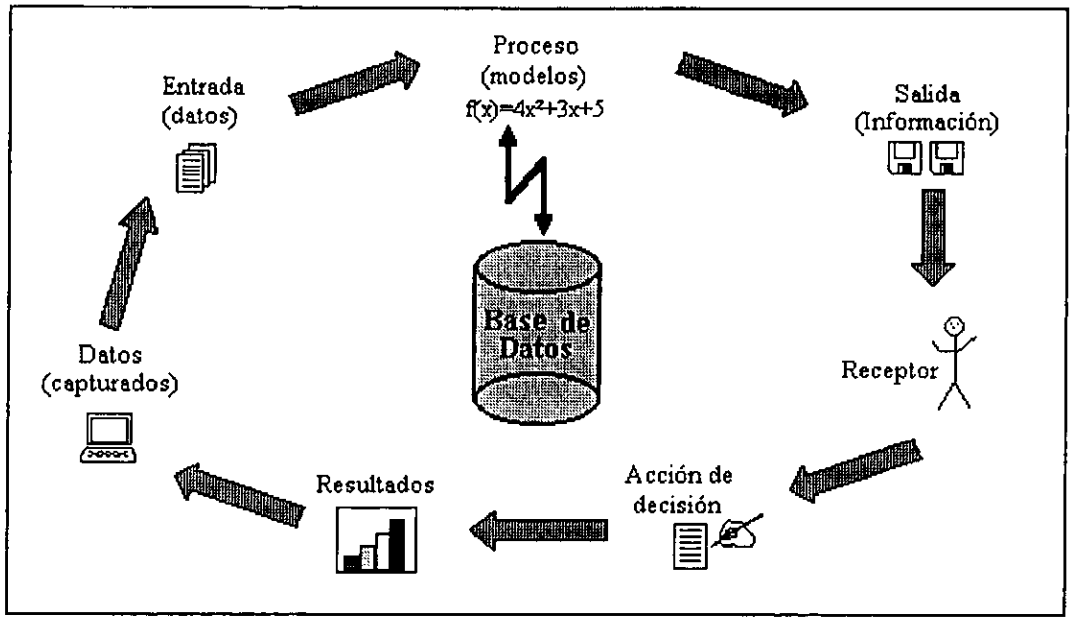


Figura 1.2 Ciclo de la información

1.4.2 Necesidad de la información

Actualmente, cada día se hace más necesario utilizar y producir información en los diferentes sectores de la sociedad. En las empresas, los inversionistas necesitan información acerca de su estado financiero y sus perspectivas futuras. La banca necesita información para evaluar el desempeño y la solidez de un negocio o empresa antes de conceder un préstamo o un crédito. Las instituciones del sector público necesitan la información que les indique las actividades de áreas tanto administrativas como normativas para efectos de impuestos y reglamentación, otras necesitan de estadísticas para hacer diversos tipos de estudios y poder implementar nuevos programas. Los sindicatos están interesados en las actividades y las utilidades de las empresas para las que trabajan sus afiliados. En el sector educativo se necesita accesar y divulgar información actualizada sobre diferentes áreas del conocimiento. Como se puede apreciar es innegable que la información forma parte de nuestra vida cotidiana.

1.4.3 Atributos de la información

La información tiene valor en la medida en que cumple con los siguientes tres atributos principales:

- **Exactitud**
- **Oportunidad**
- **Relevancia**

Exactitud.- es algo más que saber que uno más uno es igual a dos, significa que la información este libre de errores, que la información es clara y que refleja adecuadamente el sentido de los datos en los que se basa. Transmite una imagen clara al receptor, lo cual puede representarse en forma gráfica o en forma tabular. Por ejemplo, los datos de un censo.

Oportunidad.- el hacer llegar la información en un marco de tiempo es otro atributo clave de la calidad de la información. La oportunidad en la información significa que los receptores la puedan obtener cuando la necesitan.

Relevancia.- es el otro atributo clave de la información, significa que ésta responde de manera especifica al receptor sobre el ¿qué?, ¿por qué?, ¿dónde?, ¿cuándo?, ¿quién? y ¿cómo?. Cabe mencionar que lo que es relevante para un receptor no lo es para otro.

1.5 ¿Qué es un sistema de información ?

Consiste en toda una infraestructura de hardware, software, personas, procedimientos y datos desarrollada con un solo fin, el manejo de la información de manera eficiente, rápida y actualizada.

En la figura 1.3, se muestra como se combinan estos elementos para crear un sistema de información con capacidad de procesamiento de datos para un departamento o para todos los departamentos de una empresa, con el fin de proporcionar apoyo a la toma de decisiones para la gerencia.

Los sistemas de información pueden clasificarse por su nivel de complejidad y por su nivel de predictibilidad como se vio anteriormente, sin embargo, también se pueden clasificar por el área de aplicación para la que fueron creados y por el tipo de servicio que prestan. Así, por ejemplo, tenemos en cuanto al área de aplicación a los sistemas de información geográfica (GIS), los sistemas de procesamiento de imágenes o los sistemas expertos basados en el conocimiento o los sistemas de información gerencial (MIS) , y en cuanto al servicio que prestan

se puede mencionar por ejemplo, a los sistemas de consulta bibliográfica, sistemas bancarios, sistemas de reservaciones aéreas y en general todos los sistemas en línea[11] que prestan un servicio de cómputo a un determinado número de usuarios.

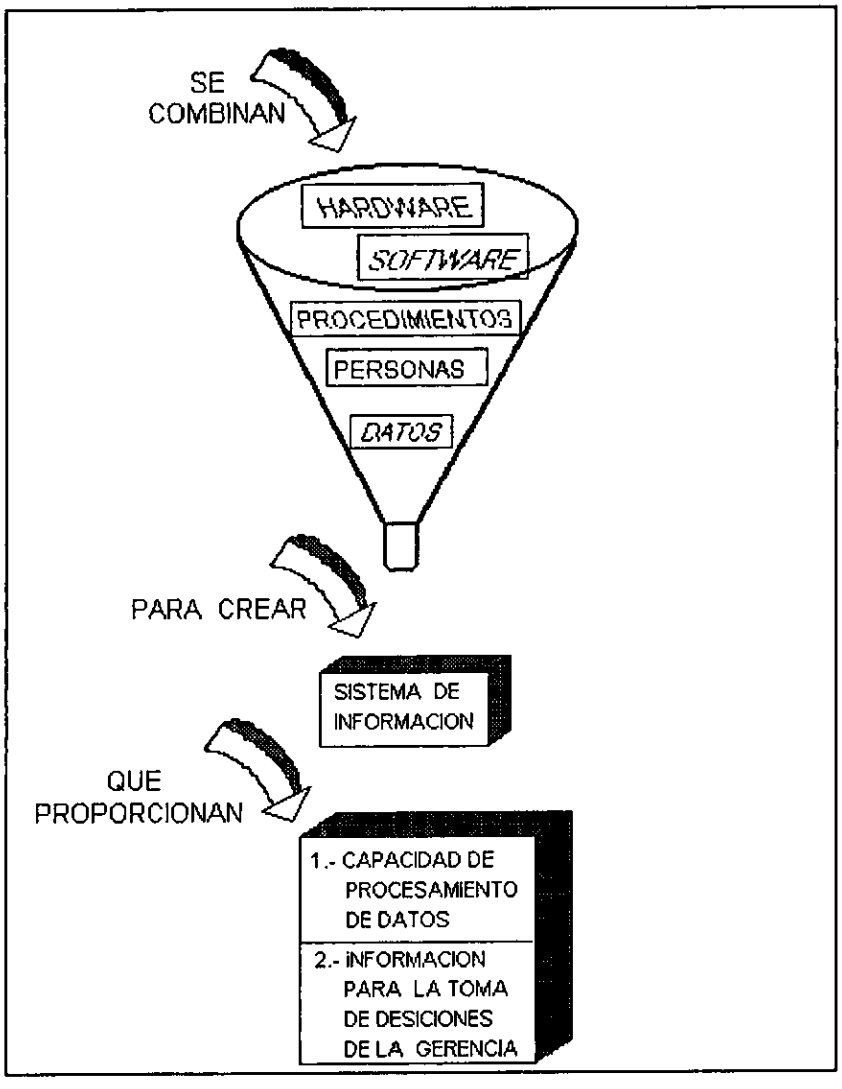


Figura 1.3 Elementos que se combinan para la creación de un sistema de información

[11] Un sistema en línea es aquel que acepta datos de entrada directamente del área donde se crearon y la salida o resultado del proceso de cómputo lo devuelve directamente a donde se requirió en cuestión de unos cuantos segundos

1.5.1 Componentes estructurales

Cualquier sistema de información, ya sea grande o pequeño, simple o complejo y sin importar a las organizaciones a las que sirven, ni la forma en que se diseñan y desarrollan, están compuestos por seis componentes estructurales: **entrada, modelos, salida, tecnología, base de datos y controles**[12].

Como se ilustra en la figura 1.4, dichos componentes estructurales pueden tomar diferentes formas, valores y contenido; pueden soportar sistemas bien diseñados; otros pueden soportar sistemas diseñados deficientemente; algunos pueden ser sencillos, otros altamente sofisticados. El que tan bien se combinen y el sistema de información que resulte depende del diseñador, el cual viene siendo el arquitecto de los sistemas. Así como un niño con un estuche para construcción de juguetes que contiene unos cuantos componentes básicos, puede construir con una buena dosis de imaginación aviones, carritos, edificios, molinos de viento, puentes, robots y hasta dinosaurios, un diseñador de sistemas puede construir sistemas de información funcionales, que satisfagan las necesidades de las empresas y de sus usuarios. La comprensión de estos componentes y sus relaciones nos dan los conocimientos básicos para describir, diseñar y desarrollar sistemas de información.

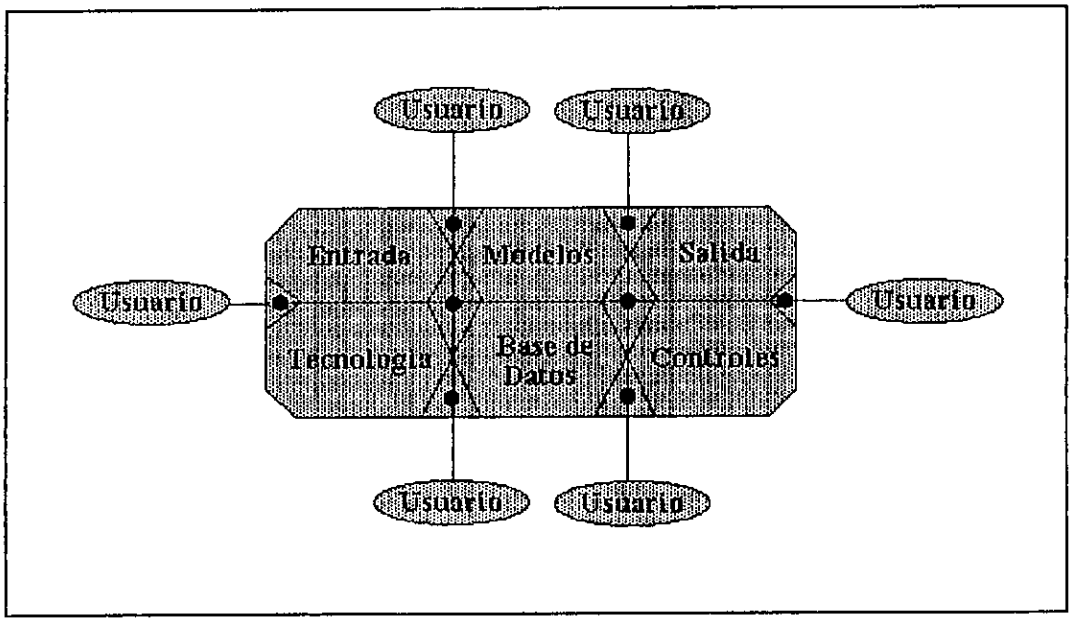


Figura 1.4 Componentes estructurales de los sistemas de información

[12] Burch, John G. *Diseño de sistemas de información, teoría y práctica*, ed. Limusa, México, 1992. pp. 57-59.

Entrada.- Representa a todos los datos, texto, voz e imágenes que entran al sistema de información así como los métodos y medios empleados para capturarlos e introducirlos. Por lo general, la entrada sigue un protocolo y un formato para que el procesamiento sea adecuado. La introducción de los datos al sistema puede ser mediante documentos, teclados, escanners, tabletas digitalizadoras, voz, pantallas sensibles al tacto (touch screen), etc.

Modelos.- Manipulan la información de entrada y/o los datos almacenados en una base de datos, para producir los resultados deseados o salidas. Los modelos son una forma de abstracción o representación de un fenómeno de la realidad. El analista de sistemas construye diversos modelos con diferentes propósitos, en el caso de un sistema de información para la administración, un modelo puede ser el desarrollo de un conjunto de procedimientos que debe seguir el personal para realizar ciertas tareas, otro propósito puede ser el desarrollar modelos lógico-matemáticos para convertir los datos en información. Por ejemplo, un tabulador de números que representan las edades de una población (datos), se manipulan sumando las edades y dividiendo la suma entre el total de habitantes de la población para obtener la edad promedio (información).

Salida.- Es el producto de un sistema de información, éste puede estar representado por reportes para el nivel de gerencia que sirven para apoyar la toma de decisiones, o para todos los usuarios dentro e incluso fuera de la organización. La salida, debe cumplir con las características de información de calidad descritas anteriormente (exactitud, oportunidad y relevancia). La salida es en general el componente que guía e influye en los otros componentes.

Tecnología.- Es otro componente de un sistema de información y podemos verlo como la "caja de herramientas" que hace el trabajo pesado en el sistema, es decir, por medio de la tecnología empleada se captura la entrada, se activan los modelos, se almacenan y se accesan los datos, se produce y transmite la salida y en general ayuda a controlar todo el sistema. La tecnología consta principalmente de tres componentes:

1) La computadora y los medios de almacenamiento.

2) El software, y

3) Las telecomunicaciones.

Las telecomunicaciones comprenden los medios para transmisión de datos entre terminales o computadoras remotas, estos medios pueden ser microondas (permiten el manejo simultáneo de miles de canales), líneas telefónicas o fibra óptica.

El software corresponde a los programas que dan las instrucciones sobre la forma de procesar los modelos y hace que funcione el hardware de la computadora.

El hardware esta constituido por una variedad de dispositivos que proporcionan el soporte físico para los demás componentes del sistema de información.

Base de datos.- La base de datos es un componente muy importante de cualquier sistema de información ya que es el lugar donde se almacenan los datos necesarios para atender las necesidades de todos los usuarios. Los datos están almacenados en forma de archivos que se pueden ligar de diferentes maneras, además pueden ser como se mencionó anteriormente, texto, voz, imágenes, números o una combinación de éstos.

Se tiene que ver a la base de datos desde dos puntos de vista, el físico y el lógico, el primero comprende los medios de almacenamiento, como las cintas, discos duros, discos flexibles, discos compactos (CD's) y demás dispositivos donde se almacenan los archivos de datos realmente, y el segundo es el referente a la forma de asociar y recuperar los datos almacenados para satisfacer las necesidades específicas de los usuarios. Si una base de datos esta estructurada correctamente podemos asegurar la recuperación de información de calidad, es decir, información exacta, oportuna y relevante.

Controles.- Este componente esta constituido por una diversidad de medidas de control para asegurar la protección e integridad de los datos y la operación uniforme del sistema. Ya que cada programa que accesa a una base de datos representa una amenaza potencial para la integridad de la base de datos.

1.5.2 La complejidad inherente del software

Existe un problema básico por el cual surge la necesidad de adoptar nuevas técnicas para desarrollar sistemas de software, dicho problema es la inherente y creciente complejidad del Software.

Evidentemente existen algunos sistemas de software que no son complejos, como por ejemplo, las aplicaciones que son especificadas, construidas, mantenidas y usadas por la misma persona. Con esto no quiero decir que tales sistemas sean imperfectos o mal hechos, ni mucho menos restarle méritos a sus creadores. Sino que tales Sistemas tienden a tener propósitos muy limitados y un corto período de vida.

Así como también existen grandes sistemas de software, o lo que algunos autores llaman "Industrial-Strength Software" (Software industrial robusto o potente). Una de la característica de este tipo de sistemas es que muestran complicadas interelaciones entre los componentes que lo integran, ejemplos de este tipo de aplicaciones son los sistemas que mantienen la integridad de miles de registros de información mientras permiten actualizaciones concurrentes y queries (consultas); y sistemas para el manejo y control de entidades del mundo real, tal como el control del tráfico aéreo o de ferrocarriles. Tales sistemas tienden a tener

un largo período de vida, muchos usuarios incluso, llegan a depender del propio funcionamiento de tales sistemas.

Otra característica distintiva de este tipo de Software es que es muy difícil, si es que no imposible, para el desarrollador individual el comprender todas las sutilezas de su diseño, por lo que es necesario contar con equipo de desarrollo para este tipo de aplicaciones.

La complejidad de la que se esta hablando parece ser una propiedad inherente de todos los grandes sistemas de Software. En esencia lo que quiero decir es que podemos entender esta complejidad, pero nunca podemos hacer que desaparezca. Ahora bien se propone el modelo de orientación a objetos como la técnica que nos ayude a manejar la complejidad de la que se habla.

Las principales ventajas de los modelos orientados a objetos descansan en la posibilidad de hacer frente a dos temas esenciales en la Ingeniería de Software; como son el manejo de la complejidad y el mejoramiento de la productividad en el proceso de desarrollo. La programación orientada a objetos guía estos temas fomentando las siguientes estrategias de desarrollo de software:

- escribir código reutilizable
- depurar módulos de código existentes, y
- compartir código con otros sistemas.

De esta manera al reutilizar código de alta calidad, la complejidad se reduce y la productividad se mejora. Entre los mecanismos orientados a objetos que nos ayudan para la reutilización de código, particularmente tenemos la *herencia*. En lugar de copiar y modificar módulos como generalmente se hace en la programación estructurada, los programadores pueden utilizar librerías de clases que contengan código depurado y comprobado.

Por otro lado, frecuentemente la complejidad de los sistemas de software toma la forma de una jerarquía natural, es decir, un sistema complejo esta compuesto de subsistemas interrelacionados que tienen sus propios subsistemas, y éstos a su vez pueden contener a otros subsistemas y así hasta alcanzar el nivel más bajo, que correspondería a un componente elemental.[13] Este hecho facilita el entendimiento y la descripción de los sistemas complejos y de sus partes permitiendonos estudiar cada parte de una manera relativamente aislada.

---

[13] Grady, Booch. *Object Oriented Design with Applications*, The Benjamin Cummings Pb. U.S.A. 1991. p.8.

1.5.3 Breve historia del desarrollo de sistemas

El desarrollo de un sistema de información, independientemente de su tamaño y de su complejidad, requiere de un conjunto de actividades coordinadas y del empleo de algunas herramientas y modelos. La metodología de desarrollo de sistemas es una forma estándar de coordinar esas actividades. Se han observado diferentes metodologias para construir sistemas a lo largo de la historia del desarrollo de los mismos.

En los inicios del desarrollo de sistemas basados en computadoras, el problema era considerado como un problema de programación, al principio se escribía el código en lenguaje máquina o ensamblador, años después en COBOL, PL1, BASIC, etc. Se producía gran cantidad de código el cual era difícil de depurar, no estaba documentado, era difícil cambiarlo, los programas resultantes muchas veces fallaban o no satisfacían los requerimientos de la empresa.

La tendencia de los programadores poco a poco fue hacia la producción de software de mejor calidad. La búsqueda de un mejor enfoque condujo a la noción de la programación estructurada.

La filosofía de la programación estructurada, esta enfocada a administrar la complejidad mediante el establecimiento de una metodología de programación descendente (top-down), es decir, el programa se divide en módulos independientes y más pequeños; y los componentes del mismo se organizan en una estructura jerárquica. La programación estructurada permitió dar un gran paso en el diseño de sistemas, pues la técnica de descomposición funcional permite atacar cada uno de los puntos claves para resolver un problema.

Uno de los problemas que se pueden presentar al utilizar la programación estructurada es que los programadores tienden a olvidar cual es el problema global que se desea resolver, concentrándose demasiado en como implementar las funciones del programa. Esto trae como consecuencia, código demasiado específico para resolver un caso particular.

A medida que el tamaño y la complejidad de los sistemas de software aumenta, también lo hacen las interrelaciones entre los componentes del mismo. En la presente década las aplicaciones necesitan satisfacer exigencias cada vez mayores, utilizar arquitecturas y estructuras de datos más complejas y ser utilizadas por una base de usuarios cada vez mayor. Todos estos factores han llevado a la necesidad de dar un salto en la capacidad de los desarrolladores de sistemas para construir, ampliar, y mantener sistemas grandes y complejos.

Es precisamente este uno de los problemas básicos por el cual surge la necesidad de adoptar nuevas técnicas para desarrollar sistemas de software que sean más flexibles y fáciles de usar. La orientación a objetos proporciona un nuevo paradigma para la creación de sistemas de software. En este nuevo paradigma,

los **objetos** y las **clases** son los pilares, mientras que los **métodos**, los **mensajes** y la **herencia** proporcionan los mecanismos básicos.

La Programación Orientada a Objetos marcó un paso hacia una nueva metodología de programación, de igual importancia que la programación estructurada. Al igual que ella no es fácil dar una clara definición de su significado. La programación orientada a objetos no es del todo nueva, surgió a finales de los años 60 con la propuesta del lenguaje de programación **SIMULA 67**, desarrollado por Kristen Nygaard y Ole-Johan Dahl[14]. SIMULA 67 introdujo por primera vez los conceptos de clases, corrutinas y subclases, muy parecidos a los lenguajes orientados a objetos actuales. Para ese tiempo, el impacto de los nuevos conceptos no fueron muy claros para la comunidad científica sino hasta el desarrollo del lenguaje **SMALLTALK** en la década de los años 70 por los científicos del Centro de Investigación Palo Alto de Xerox (Xerox PARC), el cual fue muy importante para la implementación del paradigma de la programación orientada a objetos. De hecho **SMALLTALK** se considera el más puro de los lenguajes de programación orientados a objetos, su influencia se aprecia en diversos lenguajes como son: Loops, Objetive-C, Object Pascal, Flavors, C++ y otros más.

En los años 80, el **lenguaje C** se convirtió en un lenguaje de desarrollo muy popular, años mas tarde, Bjarne Stroustrup de los laboratorios Bell de AT&T, amplió el lenguaje para crear **C++**, un lenguaje que soporta la programación orientada a objetos.

En la presente década, muchas casas productoras de software están escogiendo las técnicas de orientación a objetos para el análisis, diseño y programación de sus sistemas nuevos. Este cambio se hace con la esperanza de producir software de mayor calidad basados en que los métodos de orientación a objetos proporcionan una mejor abstracción de los datos y un mejor "ocultamiento de la información", permiten concurrencia más directamente y responden mejor a los cambios del mundo real modelado por el software.[15]

La mayor parte de lo escrito sobre las técnicas orientadas a objetos se basa precisamente sobre los lenguajes de programación orientados a objetos. El análisis y diseño se han dejado en un segundo término, sin embargo se les debe dar la importancia que merecen y deben ser pensados en forma independiente de los lenguajes de programación.

---

[14] Winblad, Ann L. *et al. Software Orientado a Objetos*, Addisson Wesley Iberoamericana. México, 1993. p. xiv.

[15] Booch, Grady. *Object-Oriented Development*, IEEE Trans. Software Engineering, Feb 1986, pp. 211-221. Tomado del artículo *Una panorámica de la Ingeniería de Software*, en la revista: Soluciones Avanzadas, marzo-abril de 1994. p. 6.

1.6 Ciclo de vida del desarrollo de sistemas

El ciclo de vida es un concepto unificador fundamental en la ingeniería de software, que describe la secuencia de pasos durante el desarrollo y mantenimiento de los sistemas de software.

El ciclo de vida es un proceso de fases múltiples que comienza con la definición de un problema y continua hasta el envejecimiento, reemplazo o destrucción del sistema. Las actividades que se realizan en cada fase están divididas según el punto de vista del analista, es decir, algunos consideran más actividades, otros las unen en una sola o suprimen algunas que consideran implícitas en otras.

Sin embargo, lo que es claro es que todo proyecto de desarrollo debe pasar por un ciclo de análisis, diseño, e implementación.

En algunas empresas, generalmente medianas, los proyectos de desarrollo de sistemas de software surgen de conversaciones entre un usuario y el analista de sistemas, quien por lo general suele ser al mismo tiempo el diseñador y el programador del sistema que se pretende desarrollar en la empresa. En estos casos el sistema nace desde el análisis, el diseño y la implementación sin mayores problemas.

En cambio, en organizaciones más grandes no sucede así, pues todo se realiza de una manera mucho más formal, es decir, la necesidad de desarrollar un nuevo sistema o modificar uno existente, se comunica por medio de un jefe de sección o departamento a la administración, para su posterior solicitud de estudio de factibilidad al departamento de sistemas, todo esto se realiza por escrito y además los integrantes de la empresa sobrentienden que el proyecto tiene que pasar por diversas fases antes de completarse.

1.6.1 Diferentes modelos de ciclo de vida

A continuación se hace una breve exposición de los diferentes modelos de ciclo de vida para el desarrollo de sistemas. Se incluyen los siguientes[16]:

- Ciclo de vida clásico
- Ciclo de vida semiestructurado
- Ciclo de vida estructurado
- Ciclo de vida de prototipos

[16] Los modelos de ciclo de vida presentados en esta sección fueron tomados de: Yourdon, Edward. *Análisis Estructurado Moderno*, ed. Prentice Hall, México, 1993, pp. 89-113.

1.6.2 Ciclo de vida clásico

Se usó principalmente al inicio de la década de los años 70, se caracteriza básicamente por dos aspectos: una implementación ascendente del sistema y un comportamiento lineal y secuencial de una fase a la siguiente, esto implica que las fases se suceden secuencialmente una tras otra, debido a esta característica también se le conoce como *ciclo de vida de cascada*. Esta idea puede confundirnos con la primer característica que mencione anteriormente, sin embargo, la implementación ascendente se refiere a que los programadores llevan a cabo primero la prueba de los módulos, luego las pruebas de los subsistemas y finalmente la prueba del sistema mismo. En la figura 1.5 se muestra el ciclo de vida clásico de un proyecto.

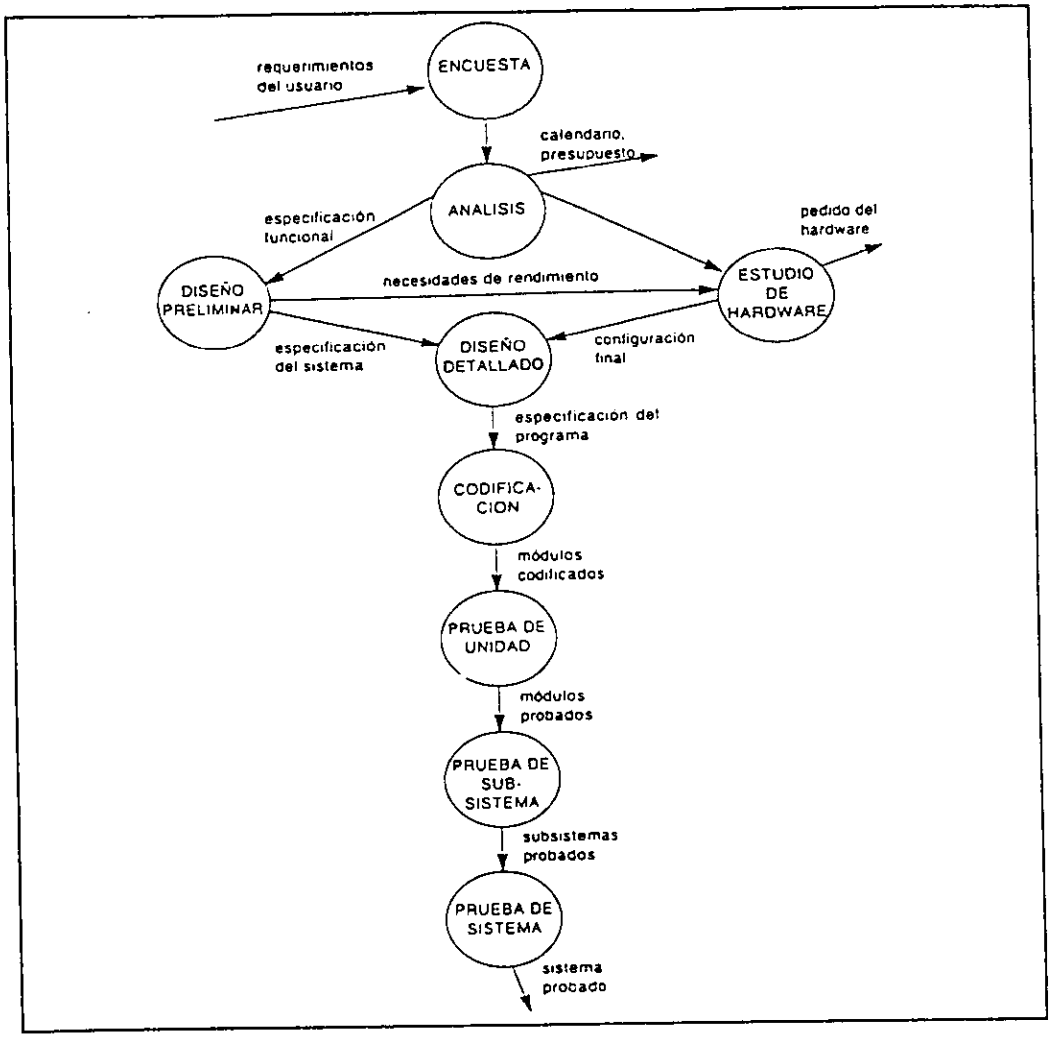


Figura. 1.5 Ciclo de vida clásico de un proyecto

Algunas de las dificultades que presenta este enfoque ascendente se presentan cuando un proyecto se atrasa y la fecha límite de entrega cae precisamente en

medio de la fase de prueba del sistema, no hay nada más que mostrarle al usuario sino una pila de listados de programas, los cuales no le ofrecen nada útil.

La eliminación de fallas suele volverse muy difícil durante la etapa final de prueba del sistema. Cuando esto sucede en un proyecto que se esta implementando de manera ascendente, es extremadamente difícil determinar en que módulo esta ocurriendo la falla, especialmente cuando existen cientos de ellos en el sistema.

El esfuerzo dedicado en cada una de las fases del ciclo de vida clásico normalmente era mayor durante las fases de codificación y pruebas del sistema constituyendo alrededor de un 65% de esfuerzo, y en las fases de análisis y diseño se dedicaba alrededor de un 35%[17]. Esta distribución daba como resultado sistemas mal especificados y de alto costo.

En la figura 1.6 se presenta la distribución del esfuerzo dedicado en cada una de las fases del ciclo de vida clásico, resaltando con gris el énfasis en las últimas fases del desarrollo.

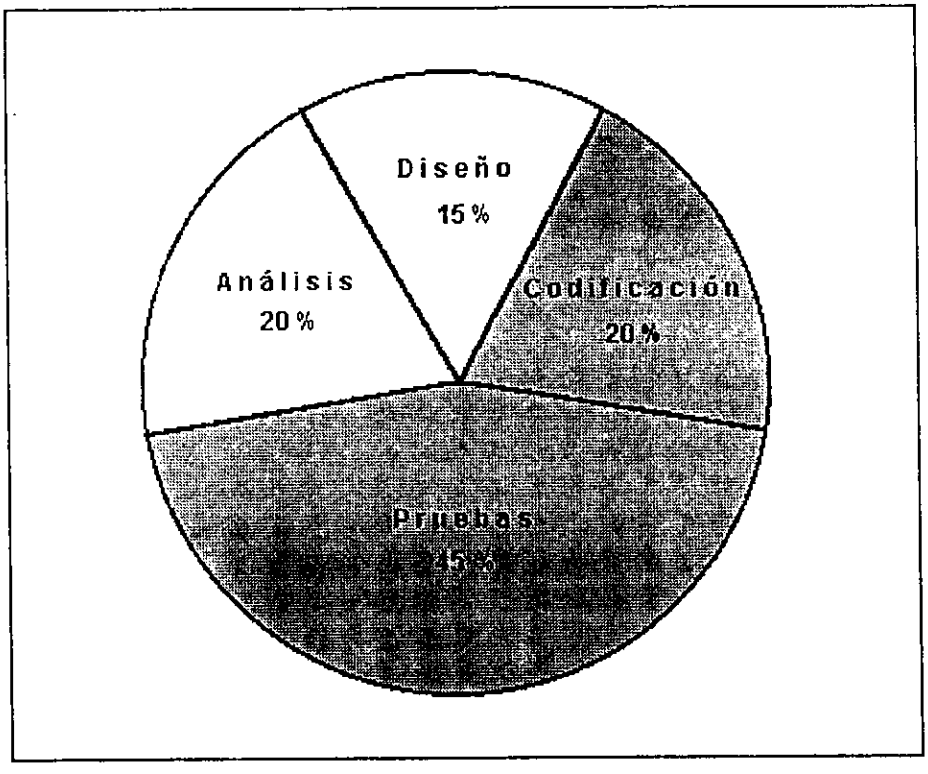


Figura 1.6 El ciclo de vida clásico del software enfatizaba las últimas fases del desarrollo con un 65% del esfuerzo en las fases de codificación y de las pruebas.

[17] McClure, Carma. *CASE la automatización del software*, ed. Addison Wesley Iberoamericana, S.A., E.U.A. 1993. pp. 250.

1.6.3 Ciclo de vida semiestructurado

Desde fines de los años 70 y principios de los 80 la tendencia fue considerar al diseño estructurado, la programación estructurada y la implementación descendente como parte del ciclo de vida de un proyecto de programación.

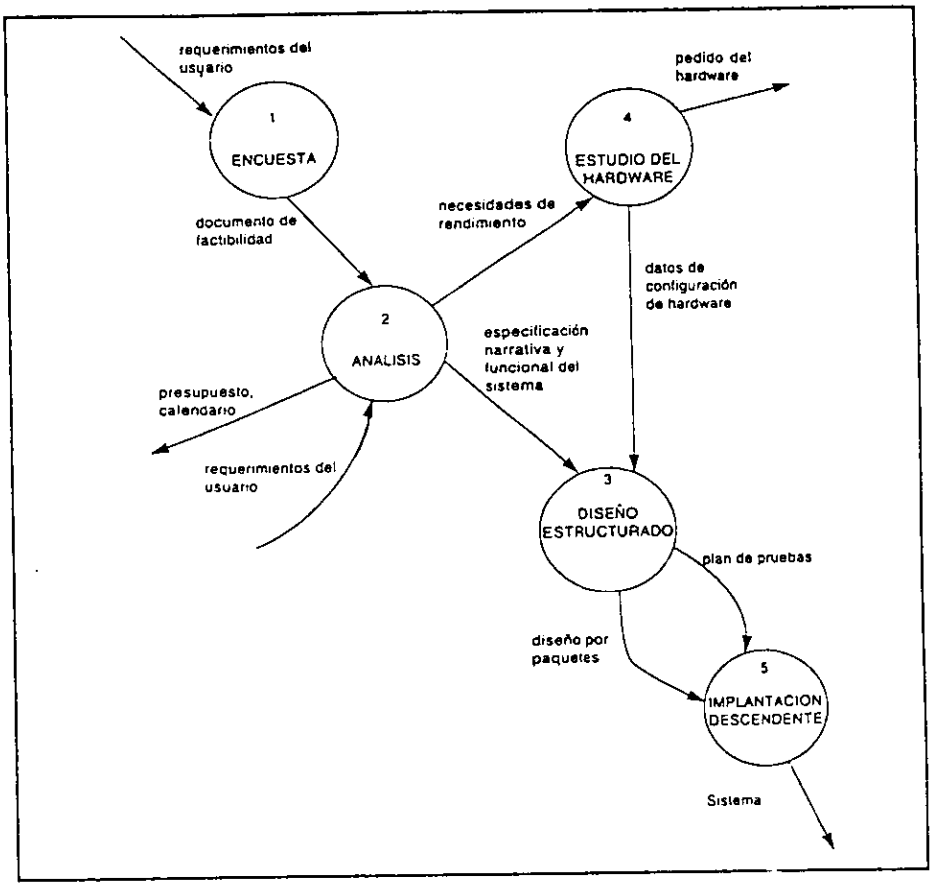


Figura. 1.7 Ciclo de vida semiestructurado

En la figura 1.7 se muestran dos detalles no presentes en el ciclo clásico:

- Se pasa de una implantación ascendente (Botton-Up) a una descendente (Top-Down)
- El diseño clásico se reemplaza por el diseño estructurado.

Al tener una implantación descendente se codifican y se prueban primero los módulos de alto nivel y luego los de nivel inferior o más detallados, la acción de ejecutar paralelamente parte de la codificación y de las pruebas, contrasta con el enfoque del ciclo de vida clásico y además puede darse la retroalimentación entre el proceso de implementación y el de análisis al fomentarse la comunicación entre el desarrollador y el usuario final.

Una parte significativa del trabajo que se lleva a cabo en la actividad del diseño estructurado es en realidad un gran esfuerzo por depurar las especificaciones erróneas.

En la figura 1.8, la actividad 3.1 representa el trabajo desempeñado por los diseñadores, es decir, la traducción de un conjunto de especificaciones en un paquete de diagramas de flujo de datos, diagramas de entidad relación, especificaciones de proceso y un diccionario de datos.

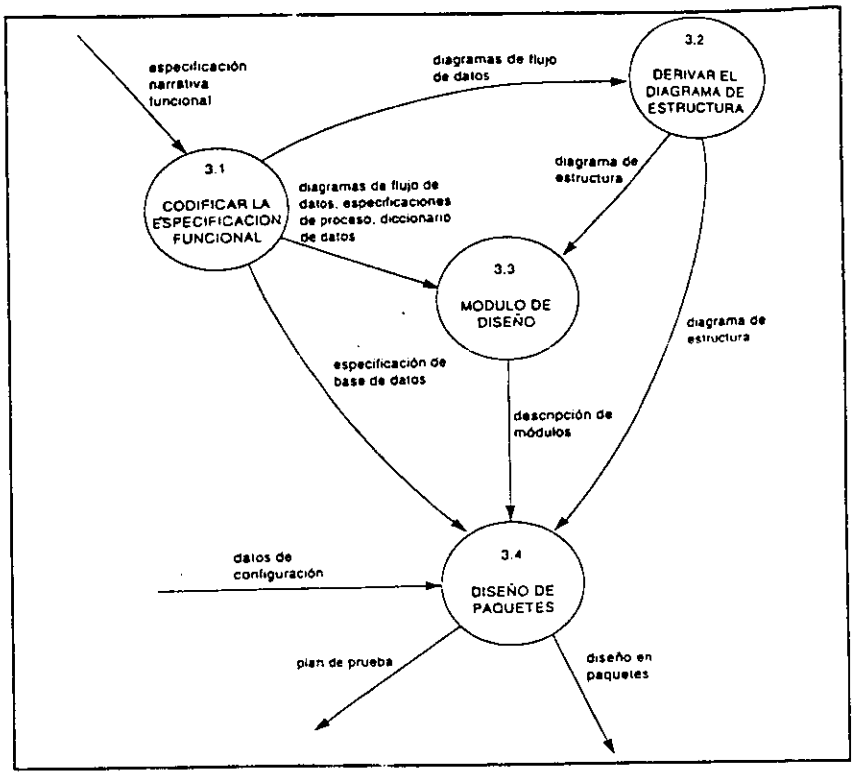


Figura. 1.8 Detalle de la actividad de diseño estructurado

1.6.4 Ciclo de vida estructurado

Dado que no es el objetivo de este trabajo analizar profundamente los modelos de ciclo de vida, resumiré las actividades de este modelo por la importancia que tiene y porque servirá como punto de referencia cuando se estudie el modelo orientado a objetos.

En el modelo, propuesto por Edward Yourdon, como se puede apreciar en la figura 1.9, la primer actividad es la *encuesta* o *estudio preliminar*, ésta se ve afectada por las políticas de los tres tipos de usuarios de un sistema:

TIPOS DE USUARIO

1. Estratégico:	Son los directivos, en general los que toman las desiciones.
2. Táctico:	Es la gente más cercana al usuario estratégico, revisa el sistema, dirige el procesamiento de datos y válida el sistema.
3. Operacional:	Tiene a su cargo el hardware y el software.

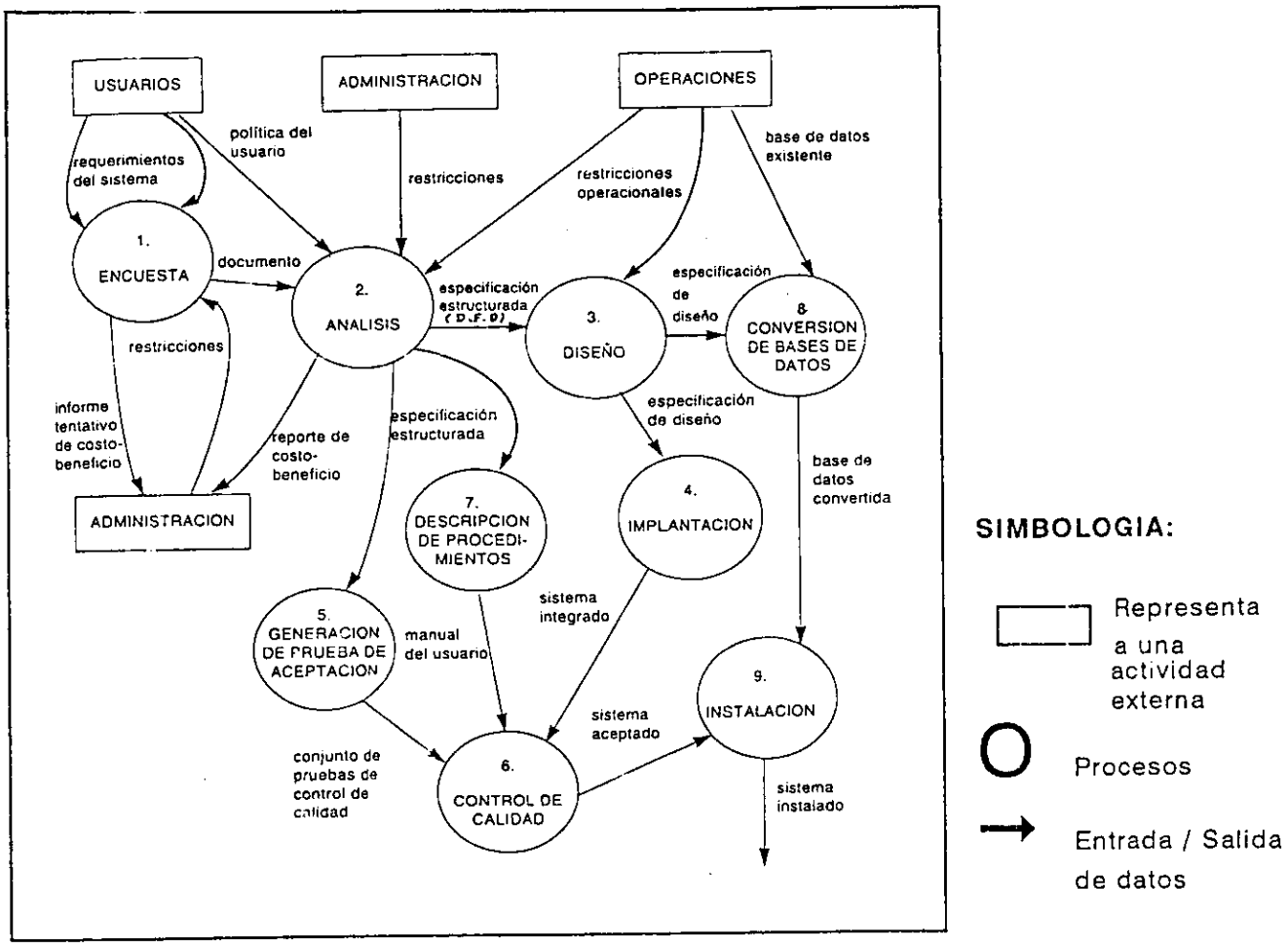


Figura. 1.9 Ciclo de vida estructurado

Los puntos que enfatiza este ciclo de vida son el análisis de los requerimientos y la especificación de diseño, debido a los problemas que llegan a causar en la implementación del sistema, ya que muchos se deben a un insuficiente análisis y una deficiente especificación de diseño.

El propósito principal del **análisis** es transformar sus dos principales entradas: las políticas del usuario y el esquema del proyecto, en una especificación estructurada; esto implica modelar el ambiente del usuario con diagramas de flujo de datos, diagramas de entidad relación, diagramas de transición de estado y algunas herramientas más de análisis.

El **diseño** consiste en convertir los diagramas de flujo de datos en un diseño modular descendente el cual permita a su vez transformar este código en algún lenguaje de programación. Además realizar el modelado de la información y el diseño de la base de datos.

La **implementación** incluye la codificación y la integración de los módulos del sistema en una estructura progresivamente más completa. Incluye así mismo programación estructurada e implementación arriba-abajo (Top-Down).

En la **generación de la prueba de aceptación** la especificación estructurada debe contener toda la información para definir un sistema aceptable desde el punto de vista del usuario.

La actividad de **aseguramiento de calidad** se conoce como prueba final o prueba de aceptación.

La actividad de **descripción del procedimiento** es la generación de una descripción formal de las partes del sistema que deben ser ejecutadas manualmente así como una descripción de como los usuarios interactuan con la parte automatizada del sistema, su resultado es el manual del usuario

La actividad de la **conversión de la base de datos** tiene lugar cuando se crea un nuevo sistema que reemplaza a uno en operación. La entrada es la base de datos actual del usuario y la especificación de diseño producida anteriormente.

La **instalación** es la fase final la cual tiene como intención poner en producción el producto, sus actividades entre otras son: entregar el hardware, el software, manuales y entrenar a la gente en la utilización del nuevo sistema.

Una parte fundamental de la estrategia de las técnicas estructuradas es dedicar el tiempo necesario a definir cuidadosamente los requerimientos y el diseño antes de pasar a los detalles de la implantación. En la figura 1.10 se muestra la redistribución del esfuerzo, de tal manera que más del 60%[18] se utiliza en las fases iniciales de análisis y diseño.

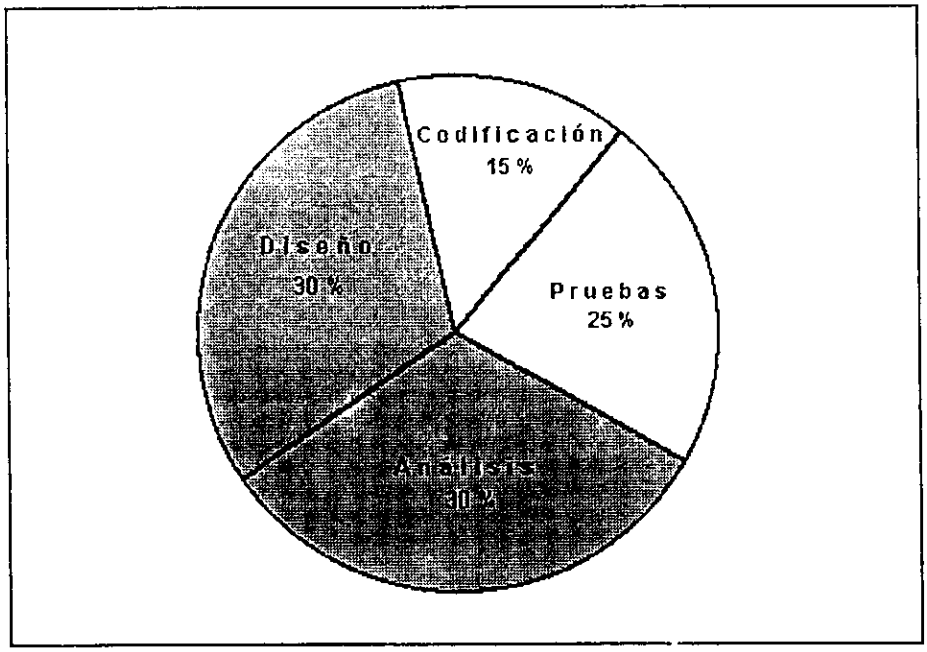


Figura 1.10 Las técnicas estructuradas pusieron más énfasis en las primeras fases del ciclo de vida con 60% de esfuerzo distribuido en las fases de análisis y de diseño.

[18] *Ibidem.* p. 251

1.6.5 Ciclo de vida de prototipos

Es una variación del enfoque descendente, se conoce como el enfoque de prototipos y lo popularizaron Bernard Boar y James Martin. Board lo describe de la siguiente manera[19]:

> "Una alternativa de enfoque para la definición de los requerimientos consistente en capturar un conjunto inicial de necesidades e implantarlas rápidamente con la intención declarada de expandirlas y refinarlas iterativamente al ir aumentando la comprensión que del sistema tiene el usuario y quien lo desarrolla. La definición del sistema se realiza mediante el descubrimiento evolutivo y gradual y no a través de la previsión omnisciente... Este tipo de enfoque se llama `de prototipos'. También se conoce como modelado del sistema o desarrollo heurístico. Ofrece una alternativa atractiva y practicable a los métodos de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales."

Como podemos apreciar el punto más importante de este enfoque es la acción de captar los requerimientos e implantarlos rápidamente, lo cual implica que la programación se inicia casi al comienzo del proyecto, con el fin de presentar al usuario una primera versión del sistema, la cual se irá refinando iterativamente a medida que el usuario vaya teniendo una mejor comprensión del sistema.

También es importante considerar que la realización de prototipos es sólo una parte del ambiente productivo de desarrollo de sistemas. Los beneficios de los prototipos dependen de las herramientas tecnológicas disponibles y del nivel de entendimiento tanto de los empleados de sistemas como de los usuarios que tienen relación con el prototipo.

Cualquier tipo de prototipo debe reducir la cantidad de documentación redundante que tiende a retrasar el desarrollo de sistemas. Un prototipo le permite al usuario revisar e interactuar con lo que el sistema hará; no hay necesidad de basarse en detalles narrativos imprecisos ni en la documentación del sistema. Sin embargo la realización de prototipos no puede reemplazar a todos los tipos de documentación, como la documentación de operación.

---

[19] Citado por: Yourdon, Edward. *Análisis Estructurado Moderno*, ed. Prentice Hall, México, 1993, pp.108.

El cuadro 1.2 resume las características de los modelos de ciclo de vida vistos hasta este momento así como sus ventajas y desventajas.

MODELO DE CICLO DE VIDA	CARACTERÍSTICAS	DESVENTAJAS	VENTAJAS
CLASICO	Fuerte tendencia a la implementación ascendente del sistema (Bottom-Up) Comportamiento lineal y secuencial de una fase a la siguiente Durante el lapso que consume el desarrollo del sistema, el usuario cambia de parecer sobre lo que quiere que haga el sistema Se apoya en técnicas obsoletas de programación, esto es, tiende a NO utilizar diseño estructurado, programación estructurada y otras técnicas modernas de desarrollo	Nada esta terminado hasta que todo esta terminado No se adecua fácilmente a los cambios del entorno del sistema (políticas, competencia, reglamentos, etc.) Los errores más simples se detectan al inicio del período de pruebas y las más graves al final (lo cual puede obligar a la recodificación de varios módulos, impactando así al calendario del proyecto) La depuración del código tiende a ser extremadamente difícil Durante las últimas etapas de prueba del sistema	
SEMI ESTRUCTURADO	Tiende a reconocer que el diseño estructurado, la programación estructurada y la implementación descendente forman parte del ciclo de vida del proyecto Es un enfoque en el cual los módulos de alto nivel se codifican y prueban primero, seguidos por los de bajo nivel, más detallados El diseño clásico se reemplaza por el diseño estructurado, que es un enfoque de diseño formal de sistemas	No toma en cuenta los cambios evolutivos, o sea, los requerimientos futuros no se contemplan No se motiva la reutilización de código Los sistemas son muy especificos Se basan en funciones muy generales	La implantación descendente permite ejecutar paralelamente algunas actividades, como parte de la codificación y de las pruebas Puede darse alguna retroalimentación entre algunas actividades Se propicia que el desarrollador tenga un acercamiento con el usuario y éste pueda expresar si desea cambiar las especificaciones

MODELO DE CICLO DE VIDA	CARACTERÍSTICAS	DESVENTAJAS	VENTAJAS
ESTRUCTURADO	Al igual que el ciclo de vida semiestructurado tiene implementación descendente (Top-Down) En el análisis se transforman sus dos principales entradas: las políticas del usuario y el esquema del proyecto, en una especificación estructurada; esto implica modelar el ambiente del usuario con diagramas de flujo de datos, diagramas de entidad relación, diagramas de transición de estado y algunas herramientas más de análisis	No motiva la reutilización de código No se adecua al desarrollo rápido de prototipos posibilitado por los lenguajes de cuarta generación (4GL)	Las mismas del ciclo de vida semiestructurado
PROTOTIPOS	Se inicia la programación casi tan pronto como se inicia el proyecto Se desarrolla un modelo funcional el cual posteriormente se descartará para reemplazarlo por el sistema final Los que desarrollan prototipos hacen uso de las siguientes herramientas de software: Diccionario de datos integrado Generador de pantallas Lenguajes de cuarta generación Lenguajes de consulta (no procedimentales), como SQL	Existe el peligro de querer convertir al prototipo en el sistema final Se puede concluir un proyecto sin dejar un registro permanente de los requerimientos del usuario Se dificulta el mantenimiento con el paso del tiempo	Reduce la cantidad de documentación redundante Un prototipo, permite al usuario revisar e interactuar con lo que el sistema hará

Cuadro 1.2 Diferentes modelos de ciclo de vida para el desarrollo de sistemas

1.7 Tendencias en el desarrollo de sistemas de software

Actualmente la historia de la orientación a objetos se caracteriza por el énfasis que se ha trasladado de la programación al análisis y al diseño. Además, la atención se ha centrado en las aplicaciones en las que se requiere de una gran manipulación de datos. Existe una fuerte tendencia a incorporar los métodos orientados a objetos, tanto en los sistemas de administración de bases de datos como en los métodos estructurados existentes y en las herramientas CASE que les dan soporte. Otra tendencia es hacia la producción de herramientas completamente orientadas a objetos. El aumento de la preocupación por los costos de mantenimiento conduce a reconocer la reutilización de código como un problema de la programación, el análisis y el diseño tradicional.

Por otro lado, la capacidad de creación de software no está a la par con la evolución del hardware ya que la capacidad del hardware rebasa a los sistemas creados hasta ahora, este puede ser un factor no fundamental pero si que debemos considerar para crear software más rápidamente y a un menor costo. Me atrevería a decir que se debe dejar de construir software "artesanal" para entrar de lleno a una era en la que construyamos software "industrial" el cual pueda construirse a partir de componentes reutilizables, lo cual conlleva un ahorro significativo en el desarrollo de un sistema.

Martin, James se refiere al cambio que se debe dar en desarrollo de sistemas como una "revolución industrial en el software"[20] y señala que es probable que éste cambio provenga de las técnicas orientadas a objetos, combinadas con el uso de herramientas CASE, generadores de código, programación visual y desarrollo basado en depósitos (repository-based development). Con el objetivo de maximizar la reutilización de código, así como de construir y almacenar objetos complejos.

Evidentemente este cambio se ha dado durante la presente década ya que las técnicas orientadas a objetos han modifican el punto de vista de los analistas de sistemas acerca del mundo, es decir, en vez de pensar en los procesos y su descomposición, ahora piensan en objetos y su comportamiento.

El análisis en términos de tipos de objetos y tipos de eventos (cambios de estado de los objetos), necesita una cierta disposición mental, pero es más sencillo y poderoso que el análisis no orientado a objetos. Algo igualmente importante es que los usuarios finales se comunican más fácilmente en términos de eventos y objetos que por medio de las construcciones del análisis estructural convencional.

---

[20] Martin, James *et. al. Análisis y Diseño Orientado a Objetos.* ed. Prentice Hall Hispanoamericana, México. 1994. pp. 3-10.

El diseño del software se realiza al volver a utilizar clases de objetos ya existentes y, si es necesario, construir nuevas clases. Al modelar un sistema, los analistas deben identificar sus tipos de objetos y las operaciones que hacen que los objetos se comporten en determinada forma. Por medio de las técnicas orientadas a objetos podemos diseñar sistemas complejos usando el principio de sencillez en que se basa la ingeniería, es decir, la gran ingeniería es la ingeniería sencilla.

1.7.1 Combinación de las técnicas orientadas a objetos con otras tecnologías de desarrollo de software

Martin, James apunta que las técnicas orientadas a objetos por si solas no proporcionan el cambio necesario sino que se deben combinar con otras tecnologías de software igualmente poderosas, algunas de ellas se resumen en el siguiente cuadro.

CASE (Ingeniería de Software Asistida por Computadora)	Las herramientas CASE utilizan representaciones gráficas en la pantalla para ayudar a automatizar la planeación, el análisis, el diseño y la generación del software.
I-CASE (CASE integrado)	Se refiere a un conjunto de herramientas CASE que soportan todas las etapas del ciclo vital, incluyendo la generación de código, y un depósito lógicamente consistente.
PROGRAMACIÓN VISUAL	Permite a los diseñadores de software introducir, comprender, reflexionar, hacer pruebas y controlar programas en la máquina mediante notaciones gráficas, con colores e incluso sonido.
GENERADORES DE CÓDIGO	Los generadores de código producen código sin errores de sintaxis en forma automática a partir de diseños, especificaciones o imágenes de alto nivel en una pantalla CASE. El código se puede generar a partir de tablas de decisión, reglas, diagramas de eventos, diagramas de transición de estado.
DEPÓSITOS Y COORDINADORES DE DEPÓSITOS	Las herramientas CASE utilizan un depósito llamado enciclopedia. En ella se almacenan los conocimientos que ayudan al analista en el proceso de creación. Las herramientas de planeación colocan la información en un depósito que puede ser utilizado por las herramientas de modelado o análisis. Estas a su vez, ponen la información en otro depósito para que la utilicen las herramientas de diseño, además estas ubican la información en un depósito destinado para la generación de código. En si un depósito es un mecanismo para la definición, el almacenamiento y la administración de la información relativa a una empresa, sus datos y sistemas. El coordinador del depósito aplica métodos a los datos del depósito para garantizar consistencia e integridad de los datos.
ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS	El analista orientado a objetos ve el mundo como objetos (con estructura de datos y métodos) y eventos que activan operaciones, las cuales modifican el estado de los objetos. El analista crea diagramas de la estructura de los objetos y de los eventos que los modifican. El modelo del diseñador es similar al modelo del analista pero con el suficiente detalle como para crear el código, el diseño y el análisis orientado a objetos.

CAPÍTULO II

CONCEPTOS FUNDAMENTALES DEL MODELO DE ORIENTACIÓN A OBJETOS PARA EL DESARROLLO DE SISTEMAS

2.1 Objetivos

Los objetivos de este capítulo son:

Introducir la terminología y conceptos fundamentales para entender el modelo de orientación a objetos.

Abordar diferentes enfoques de clasificación, los cuales proporcionan el fundamento teórico para el análisis orientado a objetos.

Plantear la importancia que tiene la identificación de las clases y objetos relevantes para un sistema, ya que esta tarea es clave en el desarrollo orientado a objetos.

2.2 Terminología y conceptos básicos

Es importante entender algunos conceptos básicos y la terminología empleada en la tecnología de la orientación a objetos, ya que ésta abarca tanto a la programación como al análisis y al diseño.

La orientación a objetos se caracteriza por los siguientes conceptos:

Objetos: Literalmente un objeto es cualquier "cosa" que representa una entidad de la realidad, sin embargo esta definición es la más básica y no sirve para nuestro propósito, así que debemos entender este concepto en el contexto del análisis y diseño, así podemos decir que un objeto es cualquier cosa real o abstracta, en el cual se almacenan *datos* y los *métodos* que manipulan dichos datos, los datos conforman los *atributos* del objeto y son los que lo distinguen de todos los demás y se almacenan en *variables de instancia* y los métodos son las operaciones que determinan como actúa el objeto cuando recibe algún *mensaje*, de hecho el conjunto de métodos proporciona el comportamiento de un objeto.

Por ejemplo, un automóvil, una persona, un avión son objetos reales, y una factura, una imagen, una reservación aérea son objetos abstractos, los atributos se conforman por las características de cada objeto, tal como color, marca, año en el caso del objeto automóvil o nombre, edad, fecha de nacimiento, para el objeto persona y las operaciones como calcular sueldo, cancelar factura, incrementar altitud, etc. son los métodos que determinan el comportamiento de dichos objetos.

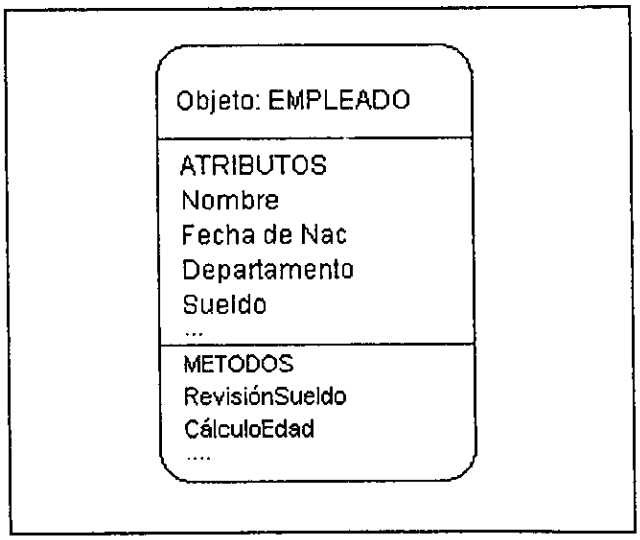


Figura 2.1 Objeto abstracto que encierra el concepto de EMPLEADO

En general, las operaciones que un objeto es capaz de realizar determinan su comportamiento o conducta, se les conoce también con los nombres de métodos o funciones miembro. En la figura 2.1 se muestra el objeto abstracto EMPLEADO, junto con algunos de sus atributos y métodos que lo definen.

Clases: Una clase es un conjunto de objetos que comparten atributos y métodos comunes. La agrupación de objetos en clases es una forma de ordenar y organizar el dominio de objetos que conforman un sistema. A un objeto que pertenece a una clase se le denomina también instancia de esa clase y tiene sus propios valores para cada atributo, pero comparte los nombres de los atributos y de sus métodos con otras instancias de la clase. En la figura 2.2 se muestra la clase **Empleado** la cual comparte los métodos Revisión_sueldo, Cálculo_edad y suprimir con todas las instancias que pertenecen a esa clase.

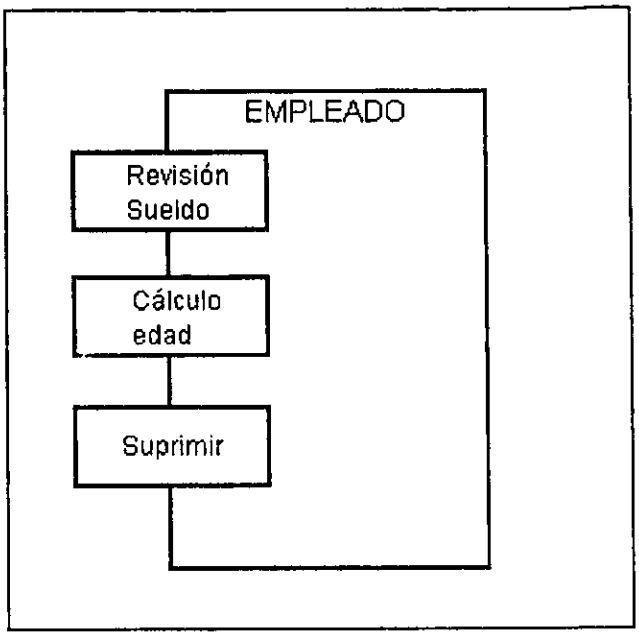


Figura 2.2 Clase Empleado con sus funciones o métodos

Métodos: (sinónimos: procedimientos, funciones, servicios), son las operaciones que son capaces de realizar los objetos de una clase. Cuándo hay herencia de clases, los métodos definidos en una superclase se heredan a los objetos de las subclases.

Mensajes: (sinónimos: llamada, activación), los objetos, las clases y sus instancias se comunican através de mensajes con el fin de eliminar la duplicación de datos. El envío de un mensaje a un objeto consiste en la activación de algunos de sus métodos. Durante la ejecución del método, el objeto no solamente tiene acceso a su estado sino que puede solicitar que se ejecuten los métodos de otros objetos, enviándoles los mensajes correspondientes. Durante la ejecución de un programa orientado a objetos se desencadena una serie de operaciones de envío de mensajes, causada por un mensaje inicial enviado a un objeto.

Encapsulado: (sinónimo: ocultamiento de información), se refiere a la práctica de incluir dentro de un objeto todo lo que necesita; estructuras de datos y métodos propios del objeto, además se hace de tal manera que ningún otro objeto necesite conocer nunca su estructura interna. Así los detalles de un objeto pueden estar ocultos dentro del mismo, de tal manera que los otros objetos, e incluso sus usuarios no puedan tener acceso no autorizado a los detalles. Ciertos datos y métodos de un objeto pueden ser privados y otros pueden ser públicos. Los primeros se refieren a la parte oculta o encapsulada de un objeto y se le conoce como su **implementación privada**; se dice que los atributos y los métodos visibles son **públicos**.

Herencia: la relación de herencia implica una estructura jerárquica del conjunto de clases de un sistema. Bajo la relación de herencia una clase puede heredar

propiedades de otra clase. A la clase que hereda se le conoce como superclase y a la clase heredera se le denomina subclase. Una subclase puede añadir y/o modificar operaciones a su comportamiento formando así una especialización de la superclase.

Una clase puede ser subclase de una o más superclases, en el primer caso se habla de herencia simple y en el segundo de herencia múltiple. En la figura 2.3(a) se ejemplifica la herencia simple, en la que cada clase hereda solo de una de mayor jerarquía las propiedades y métodos que a su vez fueron heredados, por otras de mayor jerarquía.

En la figura 2.3 (b) la clase VEHICULO ANFIBIO hereda las características de más de una clase de mayor jerarquía; VEHICULO TERRESTRE y VEHICULO MARINO, las cuales son subclases de una clase más abstracta llamada VEHICULO.

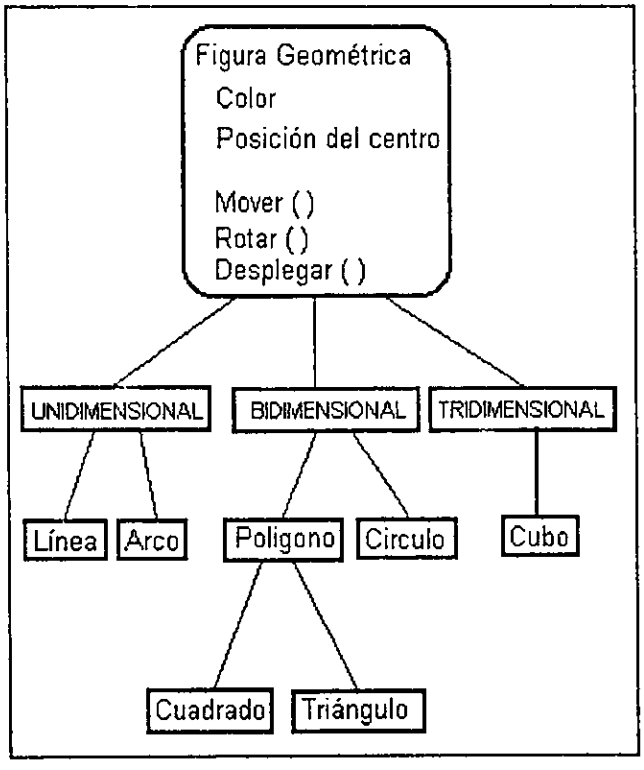


Figura 2.3 (a) Herencia simple

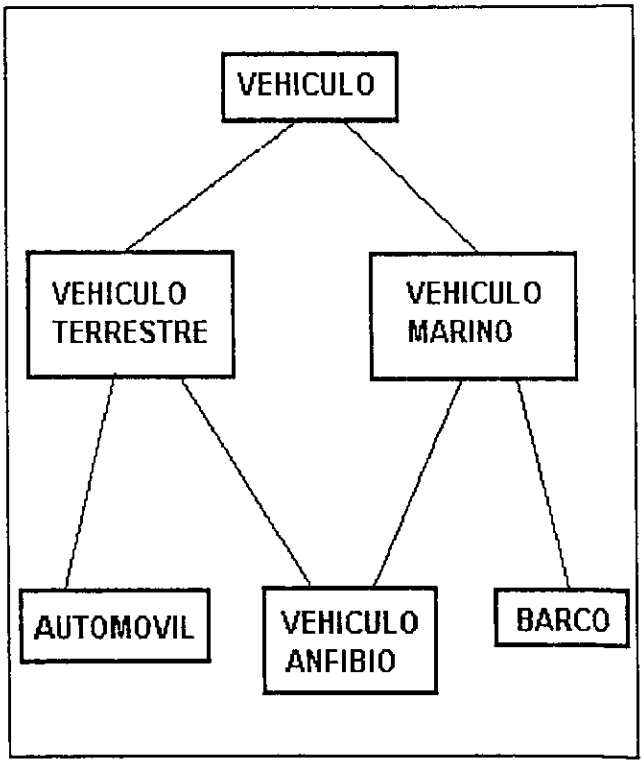


Figura 2.3 (b) Herencia múltiple

Polimorfismo. Es la capacidad de los objetos para responder de manera diferente al mismo mensaje. De esta manera el mismo mensaje puede ser interpretado de forma diferente cuándo sea recibido por distintos objetos. Con el polimorfismo un usuario puede enviar un mensaje genérico y el objeto receptor se encarga de los detalles de realización. Por ejemplo, el mensaje **imprimir** al ser enviado a una figura o imagen invocará diferentes métodos de impresión que en el caso de enviar el mismo mensaje a un objeto texto.

2.3 ¿Qué es el modelo de orientación a objetos[1]?

El modelo de orientación a objetos es una forma de construir sistemas basados en una metáfora muy poderosa llamada *objeto*, la cual representa la unidad básica de construcción. El modelo describe la estructura de un sistema en términos de las clases y objetos que lo conforman, haciendo uso de los principios de abstracción, encapsulado, modularidad y jerarquía.

2.3.1 Elementos del modelo de orientación a objetos

Se considera que hay cuatro elementos fundamentales en este modelo:

* Abstracción: denota las características esenciales de un objeto que lo diferencian de los demás tipos de objetos

* Encapsulado: el proceso de introducir en el mismo objeto los elementos de una abstracción que constituyen su estructura y comportamiento

* Modularidad: En especial para las aplicaciones grandes en las que puede haber cientos de clases, el uso de módulos es esencial para ayudar a manejar la complejidad

* Jerarquía: Una jerarquía es la clasificación u ordenación de abstracciones, la identificación de esas jerarquías en el diseño simplifica bastante la comprensión del problema. Las dos jerarquías más importantes en un sistema complejo son su estructura de clases y su estructura de objetos. Un ejemplo de jerarquías son la herencia simple y la herencia múltiple, ya que definen una relación entre clases formando jerarquías de diferentes niveles.

Si un modelo carece de cualquiera de estos elementos, entonces no es orientado a objetos.

2.3.2 Fundamentos del modelo de orientación a objetos

Los métodos de diseño estructurado utilizan los algoritmos como bloques fundamentales para construir los sistemas. De manera similar los métodos de diseño orientados a objetos han surgido para ayudar a los desarrolladores a explotar la potencia de los lenguajes de programación basados en objetos y

---

[1] El término original en inglés es **Object Model**, que se puede interpretar como el **modelo de objetos** o **modelo de orientación a objetos**

orientados a objetos, utilizando las clases y objetos como bloques fundamentales de construcción.

Los siguientes hechos han contribuido a la evolución de los conceptos de la orientación a objetos[2]:

- Avances en la arquitectura de las computadoras
- Avances en los lenguajes de programación
- Avances en las metodologías de programación, incluyendo la modularización y la ocultación de la información

- Avances en los modelos de bases de datos
- Investigación en inteligencia artificial

El hecho de que el modelo de objetos se deriva de varias fuentes, a contribuido a que la terminología usada sea un poco compleja. Por ejemplo, un programador en Smalltalk utiliza métodos, un programador de C++ utiliza funciones miembro virtuales, y un programador de CLOS utiliza funciones genéricas, lo mismo sucede en Object Pascal se habla de conversión forzada de tipos y en Ada es conversión de tipos. Toda esta terminología puede crear confusión, por eso definiremos lo que es orientado a objetos y lo que no lo es.

2.4 Programación orientada a objetos (POO)

> "La programación orientada a objetos es un método de implementación en el cual los programas están organizados como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia[3]"

Hay tres partes importantes en esta definición:

1) la programación orientada a objetos utiliza objetos, no algoritmos como sus bloques lógicos de construcción fundamentales;
2) cada objeto es una instancia de alguna clase;
3) las clases están relacionadas con otras clases vía relaciones de herencia

Un programa puede parecer orientado a objetos, pero si falta uno de estos elementos, no es un programa orientado a objetos. Específicamente, programación sin herencia es programación no orientada a objetos; le llamamos programación con tipos abstractos de datos.

---

[2] Booch, Grady en *Object Oriented Design with aplications*, Benjammin Cummings Publishing, Co. U.S.A. 1991. p.34.

[3] *Ibidem*, pp. 35-36.

2.5 Diseño orientado a objetos (DOO)

> "El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico así como los modelos dinámico y estático del sistema que se pretende diseñar[4]"

Hay dos partes importantes en esta definición:

1) El DOO propicia una descomposición orientada a objetos y es precisamente esta característica la que hace al DOO bastante diferente del diseño estructurado; es decir, el DOO utiliza abstracciones de clases y objetos para estructurar lógicamente los sistemas y el diseño estructurado utiliza abstracciones algorítmicas.

2) La utilización de una notación para describir los diferentes modelos del diseño lógico y físico de un sistema, además de los aspectos estáticos y dinámicos del mismo. La notación y descripción de éstos modelos se explicarán en el capítulo III, por el momento debemos entender que para describir todos los detalles de un sistema orientado a objetos, es necesario utilizar estos modelos ya que cada uno representa una visión diferente del mismo..

2.6 Análisis orientado a objetos (AOO)

La influencia que el modelo orientado a objetos ha tenido en esta fase inicial del ciclo de vida, ha sido en el sentido de que en el análisis orientado a objetos se enfatiza la construcción de modelos utilizando una visión del mundo orientada a objetos, a diferencia de las técnicas de análisis estructurado tradicionales que se enfocan en el flujo de los datos dentro de un sistema.

Booch define el análisis orientado a objetos de la siguiente manera:

> "El análisis orientado a objetos es un estado en el ciclo de vida del desarrollo de un sistema en el cual se examina un problema del mundo real para entender sus requerimientos desde una perspectiva de clases y objetos que se encuentran en el vocabulario del dominio del problema.[5]"

En el capítulo III se examina el propósito y las actividades asociadas al análisis orientado a objetos.

---

[4] *Ibidem*, pp. 36-37.

[5] *Ibidem*, p. 37.

Por el momento es interesante resaltar la relación que existe entre el AOO, DOO, y la POO, básicamente ésta consiste en que los productos del AOO sirven como modelos de los cuales se puede partir para definir un DOO, a su vez los productos del DOO pueden utilizarse después como anteproyectos para la implementación completa de un sistema utilizando métodos de POO.

2.7 Beneficios de la aplicación del modelo de orientación a objetos

La aplicación del modelo no significa que se deban abandonar los buenos principios y la experiencia obtenida de la aplicación de los modelos tradicionales de análisis, diseño y programación estructurada, mejor dicho es como una evolución de ellos, solamente que con un enfoque más natural y que ayuda a manejar los problemas que se derivan de la creciente complejidad del software.

Entre los principales beneficios que se obtienen tenemos:

- Ayuda a explotar la potencia expresiva de los lenguajes de programación orientados a objetos.

- Promueve la reutilización, no solo del código sino de diseños enteros, permitiendo crear marcos de desarrollo de aplicaciones reutilizables. La implementación de los sistemas orientados a objetos es por lo general más pequeña que su equivalente no orientada a objetos, esto no solo significa escribir y mantener menos código, sino que la reutilización del software también se traduce en beneficios de costos y planificación.

- El uso del modelo de orientación a objetos produce sistemas más flexibles al cambio, esto permite que tales sistemas evolucionen en el tiempo, en vez de ser abandonados o rediseñados completamente cada vez que se producen cambios importantes en los requerimientos.

Por otra parte, la aplicación del modelo de orientación a objetos implica afrontar los siguientes problemas:

1.- Identificar correctamente las clases y objetos relevantes de una aplicación; este es un problema de clasificación que se tratará en las siguientes secciones.

2.- Adoptar una notación adecuada para poder expresar el diseño de un sistema orientado a objetos. este problema es tema del capitulo III y se abordará utilizando la notación de Grady Booch.

3.- Utilizar un proceso que nos conduzca a desarrollar un sistema orientado a objetos bien estructurado. Igualmente en el capitulo III se tratará el proceso que permite un desarrollo incremental e iterativo.

2.8 Clasificación

Como se acaba de mencionar en la sección anterior, uno de los problemas que debemos enfrentar para aplicar eficazmente el modelo orientado a objetos es la correcta identificación de las clases y objetos relevantes para una aplicación dada. El problema se resuelve mediante la clasificación de objetos con características comunes, este proceso debería parecernos muy familiar, puesto que desde niños aprendimos a ordenar nuestro conocimiento de las cosas que nos rodean de acuerdo a las características que éstas presentan como son: color, forma y tamaño. En la figura 2.4 se presenta de una manera muy sencilla como la clasificación nos permite establecer agrupaciones de objetos con características comunes.

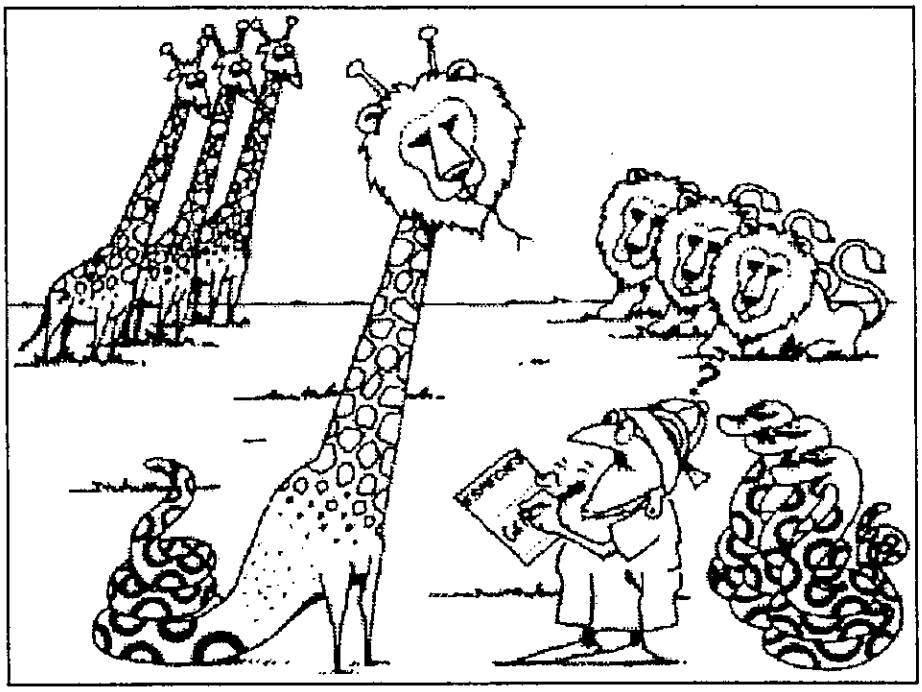


Figura 2.4 La clasificación es el medio por el cual ordenamos el conocimiento. (figura tomada de:Booch, Grady, *Object Oriented Design with aplications*, Benjammin Cummings Publishing, Co. U.S.A. 1991. p. 133.)

Sin embargo, en el contexto del análisis y diseño orientado a objetos, la identificación de clases y objetos es una tarea que conlleva cierta dificultad, porque implica dos actividades importantes:

1) El descubrimiento de las abstracciones clave que forman el vocabulario del dominio del problema.

   Mediante el descubrimiento, se reconocen las abstracciones utilizadas por el experto del dominio (muchas veces es el usuario final); si el experto habla

de ellas, entonces las abstracciones pueden ser importantes. Por ejemplo, si un cliente que utiliza un cajero automático habla en términos de cuentas, depósitos, retiros, transferencias, etc.; estas abstracciones forman parte del vocabulario del dominio del problema. Pero si se desarrolla un sistema parecido se podrían utilizar las mismas abstracciones, y también se podría introducir algunas nuevas que no son parte del dominio del problema pero son útiles para el diseño del sistema, como podrían ser: bases de datos, manejadores de pantallas, listas, colas etc.

2) La invención de los mecanismos que especifican como colaboran los objetos para proporcionar un comportamiento determinado.

   Básicamente un mecanismo es una decisión de diseño sobre como trabajan conjuntamente las clases para producir el comportamiento que satisfaga los requerimientos del problema. De esta manera los mecanismos nos pueden representar un patrón de comportamiento.

   Mientras las abstracciones clave reflejan el vocabulario del dominio del problema, se podría decir que los mecanismos son el alma del diseño, porque durante este proceso se debe considerar no sólo el diseño de las clases de manera individual, sino también como trabajan conjuntamente las instancias de esas clases, también se definen los métodos que más convengan para cada una de las clases, creando así lo que se conoce como el **protocolo de una clase**, el cual abarca todas las operaciones que se requieren para implantar el comportamiento y todos los mecanismos asociados con cada una de sus instancias.

También es importante considerar dos factores que podrían dificultar el proceso de clasificación: primero, cualquier clasificación es relativa a la perspectiva del observador que la realiza. Esto quiere decir que en un proyecto grande de software en el que existe un equipo de analistas para algunos de ellos unos objetos serán relevantes mientras que para otros no lo serán. Esta situación la he querido representar en la figura 2.5 en la cual diferentes observadores pueden clasificar de diferentes formas el mismo objeto.

El segundo factor a considerar es que una clasificación inteligente requiere una tremenda cantidad de creatividad puesto que existen objetos que a simple vista no tienen relación alguna, sin embargo, hay "algo" que los relaciona y el descubrimiento precisamente de ese "algo" es lo que sólo una mente creativa puede hacer.

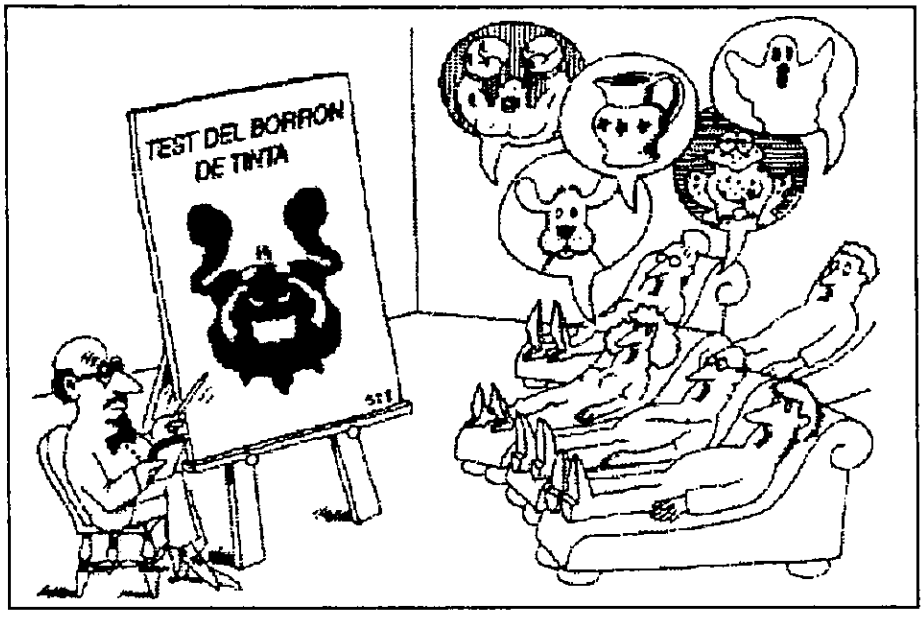


Figura 2.5 Diferentes observadores pueden clasificar el mismo objeto de diferentes formas. (Figura tomada de: Booch, Grady, *Object Oriented Design with aplications*, Benjammin Cummings Publishing, Co. U.S.A. 1991. p. 136.)

Existen tres diferentes enfoques de clasificación que proporcionan el fundamento teórico para el análisis orientado a objetos, y nos ofrecen una serie de reglas prácticas que se pueden aplicar para identificar clases y objetos en el diseño de un sistema de software complejo. Estos enfoques son los siguientes:

- Clasificación por propiedades
- Clasificación por conceptos
- Clasificación por asociación con un prototipo

2.8.1 Clasificación por propiedades (también se le conoce como categorización clásica). Todas las entidades que tienen una propiedad o un grupo de estas en común forman una categoría, se emplean criterios de similitud entre objetos. Las propiedades pueden denotar algo más que características medibles, tales como tamaño, color, forma o substancia; también abarcan comportamientos observables. Las propiedades que hay que considerar en una situación dada dependen mucho del dominio del problema. Por ejemplo, el color de un coche puede ser importante para el propósito de control de inventario en una planta armadora de autos, pero no es relevante para el software que controla los semáforos en la ciudad.

2.8.2 Clasificación por conceptos (o agrupamiento conceptual). En este enfoque, las clases se generan formulando descripciones conceptuales de ellas y después se clasifican sus entidades de acuerdo con dichas descripciones.

2.8.3 Clasificación por asociación con un prototipo (o teoría de prototipos). En este enfoque una clase de objetos se representa por un objeto prototípico (un patrón), y a un objeto se le considera como miembro de esta clase si y sólo si se parece a este prototipo.

2.8.4 Aplicación

Los tres enfoques de clasificación anteriores tienen aplicación directa en el diseño orientado a objetos. Según la experiencia de varios desarrolladores de software se identifican primero las clases y objetos de acuerdo con las propiedades relevantes del dominio del problema en particular, haciendo énfasis en la identificación de las estructuras jerárquicas y el comportamiento de las clases que son parte del vocabulario del dominio del problema.

Si usando este enfoque no se obtiene una estructura de clases satisfactoria, entonces hay que considerar la agrupación de objetos por conceptos, aquí nuestra atención se centra en el comportamiento de los objetos.

Si aún después de ambos intentos nuestra comprensión del problema no es del todo satisfactoria, entonces debemos considerar la clasificación por asociación, de esta manera agruparemos los objetos según el grado en que cada uno se parece a algún otro objeto prototípico.

2.9 Diferentes enfoques de análisis orientado a objetos

Hay que recalcar que los tres enfoques de clasificación vistos son el fundamento teórico para hacer un AOO. y están enfocados hacia una correcta comprensión de un problema en particular.

A continuación se revisan brevemente diferentes enfoques de AOO:

2.9.1 Enfoques clásicos. Se les llama clásicos porque su origen esta en los principios de la categorización clásica o clasificación por propiedades vista anteriormente.

Hay varios diseñadores de metodologías que han propuesto varias fuentes de clases y objetos, como por ejemplo Coad y Yourdon sugieren que las clases y objetos candidatos provienen por lo general de las siguientes fuentes:

* Estructuras	Relaciones "tipo de" y "parte de"
* Otros sistemas	Sistemas externos con los que la aplicación interactúa

* Dispositivos	Dispositivos con los que la aplicación interactúa
* Eventos recordados	Un evento histórico que debe ser registrado
* Papeles desempeñados	Los diferentes papeles (roles) que juegan los usuarios cuando interactuan con la aplicación
* Posiciones	Ubicaciones físicas, como oficinas y lugares importantes para la aplicación
* Unidades de organización	Grupos a los que los usuarios pertenecen

2.9.2 Análisis del comportamiento. Se centra en el comportamiento dinámico como principal fuente de clases y objetos. este enfoque se relaciona con el agrupamiento conceptual ya que se forman clases basadas en grupos de objetos que presentan un comportamiento similar.

2.9.3 Análisis de dominios. El análisis de dominios busca identificar las clases y objetos comunes a todas las aplicaciones dentro de un dominio dado, tales como mantenimiento de registro de pacientes, operaciones bursátiles, etc. Si uno por ejemplo esta a la mitad de un diseño y le faltan ideas sobre las abstracciones clave que existen, entonces un pequeño análisis de dominio puede ayudarnos, indicándonos las abstracciones clave que han sido útiles en otros sistemas relacionados con el nuestro. El análisis de dominios funciona bien porque la mayoría de los sistemas encajan en algún tipo de aplicación, existen muy pocos sistemas que sean verdaderamente únicos.

2.9.4 Fichas CRC (Clases/Responsabilidades/Colaboradores). La fichas CRC Son una herramienta de desarrollo simple pero efectiva para analizar escenarios. Un escenario es un esquema de los eventos que provocan algún comportamiento en el sistema. Las fichas CRC mejoran la comunicación entre los desarrolladores. Una ficha CRC es simplemente una tarjeta de 5x7 centímetros sobre la cual el analista escribe el nombre de una clase (en la parte superior de la tarjeta), sus responsabilidades[6] (en la mitad de la tarjeta), y sus colaboradores[7] (en la otra mitad). Se crea una ficha para cada clase que se identifique como relevante para el escenario. En la medida que el equipo de desarrollo avanza por ese escenario, puede asignar nuevas responsabilidades a una clase ya existente, agruparlas para

---

[6] Una responsabilidad denota la obligación de un objeto para proporcionar cierto comportamiento.

[7] La colaboración se refiere al proceso por el cual varios objetos colaboran para proporcionar el comportamiento deseado de una clase.

formar una nueva clase o dividir las responsabilidades de una clase y distribuirlas en otras clases diferentes.

2.9.5 Descripción informal en español. Es una alternativa que fue primeramente propuesta por Abbott[8], quien sugirió redactar en un lenguaje natural, (originalmente es en inglés pero en nuestro caso lo podemos hacer en español), una descripción del problema, y subrayar los nombres y los verbos. Los nombres representan objetos candidatos y los verbos representan operaciones candidatas sobre ellos.

El enfoque de Abbott es útil porque es simple y obliga a los desarrolladores a trabajar en el vocabulario del espacio del problema. Sin embargo no es un enfoque riguroso y no se adapta bien para sistemas mayores, solamente para sistemas triviales. La calidad de la lista de objetos y operaciones depende de la habilidad del analista para redactar, ya que el lenguaje es un vehículo de expresión terriblemente impreciso.

2.9.6 Análisis estructurado. Es otra alternativa para identificar clases y objetos, ya que utiliza los productos del análisis estructurado como vía de entrada al diseño orientado a objetos. Esta técnica es atractiva porque hay varios analistas con experiencia en análisis estructurado, y además existen algunas herramientas CASE que soportan la automatización de estos métodos.

Al utilizar el análisis estructurado, se empieza con un **modelo esencial**[9] del sistema, descrito por los diagramas de flujo de datos, diagramas de entidad-relación, de transición de estados, diccionario de datos y especificaciones de proceso. Estos diagramas conforman un modelo formal del problema para que sirva de punto de partida para identificar las clases y objetos significativos en el dominio del problema.

---

[8] Citado por: Booch, Grady. *Análisis y Diseño Orientado a Objetos con aplicaciones.* segunda edición, Addison Wesley, México, 1996. p.182.

[9] Es un modelo de lo que el sistema debe hacer para satisfacer los requerimientos del usuario, diciendo lo mínimo posible (de preferencia nada) acerca de como se implantará.

CAPÍTULO III

PROCESO DEL ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

3.1 Objetivos

Los objetivos de este capítulo son:

Introducir la notación gráfica de Booch para el análisis y diseño orientado a objetos.

Presentar el proceso de Booch para desarrollar sistemas basados en el modelo de orientación a objetos.

3.2 Introducción a la notación de Booch

El hecho de dibujar un diagrama no constituye ni análisis ni diseño, un diagrama se limita a capturar una descripción del comportamiento de un sistema (en el análisis), o la visión y los detalles de la arquitectura del mismo (en el diseño).

Un sistema se concibe única y exclusivamente en la mente del diseñador y la manera de hacer que ese diseño permanezca en el tiempo es plasmándolo en el papel, es por esta razón que es necesario utilizar una notación, sin embargo, ésta debe tener las siguientes características:

1) La notación debe ser estándar y bien definida, ya que esta característica permite al analista formular un diseño y poderlo compartir, logrando así comunicar sus ideas y desiciones a los demás miembros de un equipo de desarrollo en un proyecto de software. Esta característica es especialmente importante en grandes proyectos donde un equipo puede estar separado geográficamente. Por ejemplo, si un analista en el Distrito Federal hace el diseño general de un sistema, otro analista en monterrey no debería tener problemas para hacer el diseño detallado de los módulos para que cumplan con las especificaciones requeridas.

2) Soportar el apoyo de herramientas CASE[1] ya que éstas admiten la generación de diagramas de objetos, diagramas de clases, diagramas de módulos y diagramas de procesos, los cuales se analizarán en las siguientes secciones.

Es importante hacer notar que el hecho de utilizar una gran herramienta CASE no garantiza tener un gran diseño, puesto que los grandes diseños provienen únicamente de los buenos diseñadores. Si se es un mal diseñador y se cuenta con alguna buena herramienta CASE, lo único en que nos podrá ayudar es a crear diseños horribles mucho más rápido de lo que jamas se había podido.

Las herramientas CASE simplemente deben apoyar al desarrollador, permitiéndole concentrarse en los aspectos verdaderamente creativos del análisis o del diseño.

Un ejemplo del apoyo que se puede recibir al utilizar alguna herramienta CASE es cuando se utiliza un diagrama de objetos para mostrar un escenario con un mensaje que pasa de un objeto a otro, la herramienta puede encargarse de hacer una comprobación de consistencia, es decir, puede asegurar que el mensaje transmitido forma parte del protocolo del objeto. De la misma manera puede también realizar la comprobación de restricciones definidas para cierta clase.

---

[1] Existe una variedad de herramientas CASE que apoyan las técnicas y métodos orientados a objetos, si el lector quiere saber más acerca de ellas puede consultar: Graham, Ian. *Métodos Orientados a Objetos*. segunda edición, Adisson Wesley/Díaz de Santos. E.U.A. 1996. pp. 388-398.

Por el contrario, una herramienta CASE nunca nos dirá si es recomendable simplificar una estructura de clases existente o si nos convendría crear una clase nueva, ya que estas desiciones son tomadas única y exclusivamente por el diseñador.

La notación de Booch que se presenta en este capítulo pudiera parecer muy detallada, sin embargo, esto no significa que se deban utilizar todos sus aspectos en todos los diseños, sino que deben aplicarse solamente los elementos de la notación que sean necesarios para transmitir el significado deseado y nada más, porque sino caeríamos en una sobreespecificación de requisitos, situación que no es aconsejable, como tampoco lo es sobreespecificar una solución para un problema.

Ahora bien, puede suceder que los miembros de un equipo de desarrollo estén altamente capacitados y tengan una estrecha relación de trabajo, para ellos pudiera darse el caso de que un esquema aproximado o un boceto sea suficiente para capturar la comprensión del problema y realizar un buen diseño, sin embargo, será necesario dejar constancia de la visión arquitectónica del sistema para quienes posteriormente se ocupen del mantenimiento del mismo.

En el caso contrario, si los miembros del equipo no están lo suficientemente capacitados, o si los diseñadores y los programadores están separados geográficamente, entonces será necesario detallar más las especificaciones durante el proceso de desarrollo.

Cabe mencionar que existen diferentes notaciones y métodos orientados a objetos, sin embargo he elegido la notación y el método revisado de Booch porque es uno de los más antiguos y en él se basan un gran número de métodos y notaciones, hasta la fecha existen más de diez métodos de diseño orientado a objetos, por mencionar algunos tenemos los trabajos de: Jacobson, Rumbaugh, Coad y Yourdon, Constantine, Shlaer y Mellor, Martin y Odell, Wasserman, Goldberg y Rubin, Embley, Wirfs-Brock, Goldstein y Alger, Henderson-Sellers y otros más[2].

---

[2] Para consultar más sobre estos métodos y su notación véase Graham, Ian. *Métodos Orientados a Objetos*, segunda edición, Addison Wesley/Díaz de Santos. E.U.A. 1996. Capitulo 7 pp. 247-285.

3.3 La necesidad de tener múltiples vistas de un sistema

Para capturar todos los detalles de un sistema de software es necesario tener múltiples vistas sobre él; uno debe tomar en cuenta tanto la estructura de sus clases y objetos como el comportamiento que éstos presentan individualmente y en conjunto.

Este problema es semejante al de ver un evento deportivo tal como un partido de fútbol, en el que muchas veces para captar lo que esta sucediendo en determinada acción es necesario tener varios ángulos de cámara, pues cada uno nos revelará los detalles particulares de una acción que no podrían ser transmitidos por una sola cámara.

Así la notación de Booch consta de un conjunto de modelos, los cuales hacen la función de las cámaras del ejemplo anterior y cada uno representa una vista diferente de un sistema orientado a objetos.

Estos modelos son los siguientes:

- Modelo lógico
- Modelo físico
- Modelo estático
- Modelo dinámico

En la figura 3.1 se representan gráficamente dichos modelos, y podemos apreciar la interrelación que existe entre ellos, así el **modelo lógico** representa la estructura de clases y la estructura de objetos; el **modelo físico** representa la arquitectura de módulos y la arquitectura de procesos del sistema. Asimismo ambos representan el **modelo estático** de un sistema OO; y la parte o **modelo dinámico** del sistema lo constituyen el comportamiento que presentan los objetos y las clases.

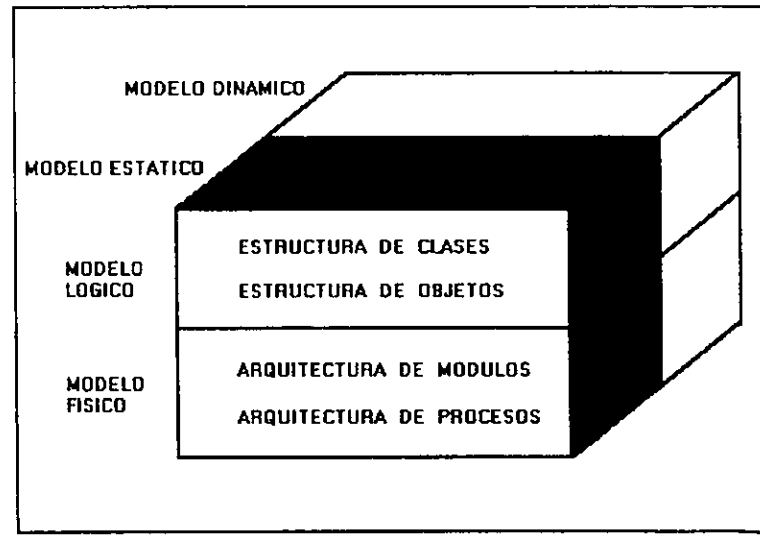


Figura 3.1 Los modelos del desarrollo orientado a objetos

Cada uno de los modelos anteriores utiliza una serie de diagramas para representar su contenido como se puede apreciar en la figura 3.2, la estructura de clases utiliza los **diagramas de clases**; la estructura de objetos utiliza los **diagramas de objetos**; la arquitectura de módulos los **diagramas de módulos**; y la arquitectura de procesos, los **diagramas de procesos**.

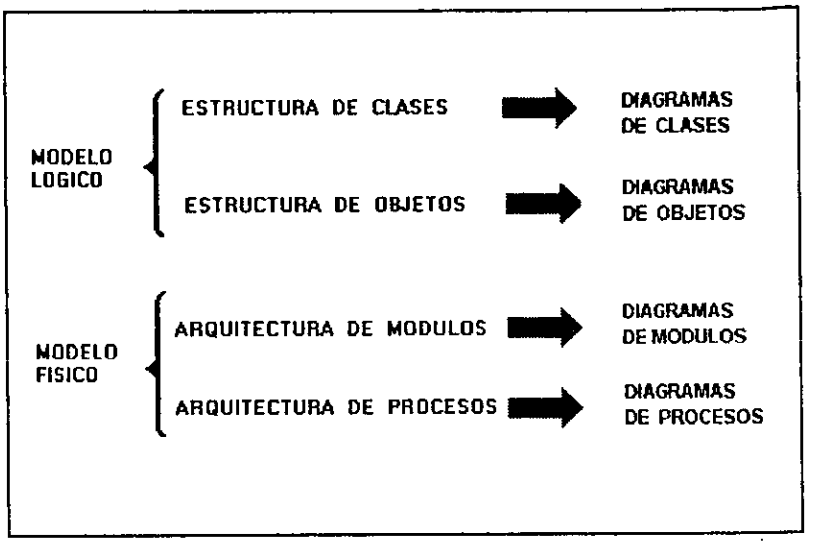


Figura 3.2 Diagramas utilizados por los modelos lógico y físico.

Conjuntamente estos cuatro diagramas nos sirven para describir la existencia y el significado de las clases y objetos así como sus interrelaciones y para definir la arquitectura del sistema.

Durante el análisis, uno se debe plantear las siguientes preguntas:

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

Con respecto a la primer pregunta se utilizan los escenarios, pues estos expresan las decisiones acerca del comportamiento de un sistema, los diagramas de objetos son el medio para describir los escenarios.

También durante el análisis se pueden utilizar los diagramas de clases para plasmar la abstracción de esos objetos en términos de sus funciones y responsabilidades comunes.

Durante el diseño; se deben responder las siguientes preguntas con relación a la arquitectura del sistema:

- ¿Qué clases existen y como se relacionan?
- ¿Qué mecanismos se utilizan para regular la forma en que colaboran los objetos?
- ¿Dónde deberían declararse cada clase y objeto?
- ¿A qué procesador debería asignarse un proceso, y para un procesador dado, como deberían planificarse sus múltiples procesos?

Se utilizan los siguientes diagramas respectivamente para responder a las preguntas anteriores:

- Diagramas de clases
- Diagramas de objetos
- Diagramas de módulos
- Diagramas de procesos

Estos cuatro diagramas son fuertemente estáticos, sin embargo, en todos los sistemas de software los eventos suceden dinámicamente, es decir, suceden dentro de un contexto de espacio y tiempo; por ejemplo: los objetos se crean y se destruyen, envían mensajes a otros objetos de manera ordenada. Es difícil describir un evento dinámico en un medio estático tal como un diagrama, pero es aquí precisamente donde cobra importancia el uso de una notación bien definida.

En el desarrollo orientado a objetos toda esta dinámica de los objetos, su estado, los eventos y la trancisión de estados se expresa mediante un modelo dinámico, es decir mediante dos diagramas adicionales:

- Diagramas de transición de estados
- Diagramas de interacción[3]

Cada clase puede tener un diagrama de transición de estados el cual nos indica el comportamiento de cada una de las instancias de esa clase y el orden en que suceden los eventos que pueden afectar el estado de cada instancia de la clase.

Un diagrama de interacción se utiliza para mostrar la ejecución de un escenario en el contexto de un diagrama de objetos. De hecho se puede utilizar conjuntamente un diagrama de objetos con un diagrama de interacción para mostrar el orden en que son enviados y evaluados los mensajes.

A continuación se describirá la notación de los diagramas mencionados.

3.4 Diagramas de clases

Se utiliza un diagrama de clases para mostrar la existencia de estas y sus relaciones en la visión lógica de un sistema. Con un solo diagrama de clases se puede representar todo o parte de la estructura de clases de un sistema. Para un sistema pequeño, puede que sea suficiente un solo diagrama de clases, pero la mayoría de los sistemas en la actualidad requieren un conjunto de tales diagramas para documentar su estructura de clases.

---

[3] El término original en inglés es timing diagrams, que se puede interpretar como diagramas de interacción o de sincronización.

Durante el análisis, se utilizan para indicar las funciones y responsabilidades que tienen en común las entidades que caracterizan el comportamiento de un sistema.

Durante el diseño, se utilizan para documentar la estructura de las clases que forman la arquitectura de un sistema.

Los dos elementos esenciales de un diagrama de clases son las clases y sus relaciones básicas.

3.4.1 Clases

Una clase se representa en un diagrama simplemente con un icono como el que se muestra en la figura 3.3. Su contorno es como el de una nube[4]; algunos autores le llaman mancha amorfa.

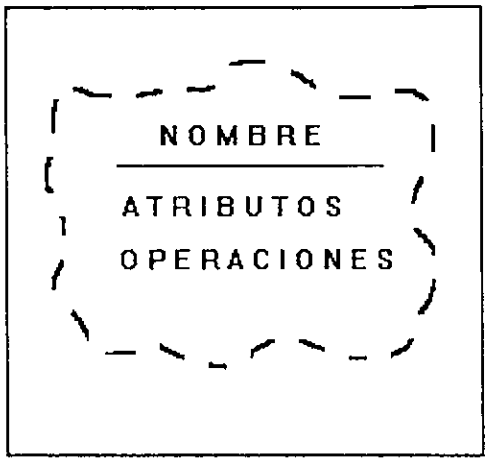


Figura 3.3 Icono para representar una clase

Se requiere asignar un nombre para cada clase; y este se coloca dentro del icono, si el nombre de la clase es muy largo se puede abreviar o bien, agrandar el icono. Cada nombre de clase debe ser único para la categoría de clases que engloba a esta.

En algunos diagramas de clases es útil escribir algunos de los **atributos** y **operaciones asociadas** con una clase. Estos representan una visión abreviada de la especificación completa de la clase, que sirve como punto de declaración para todos sus miembros. Un atributo se utiliza tanto en el diseño como en el análisis para expresar una propiedad particular de la clase.

Un atributo puede tener un nombre, una clase o ambos, y opcionalmente un valor por default como se muestra a continuación.

- **A** Nombre del atributo solamente
- **:C** Clase del atributo solamente
- **A:C** Nombre y clase del atributo
- **A:C=E** Nombre, clase y valor por default del atributo

[4] El icono en forma de nube se deriva del trabajo de Intel en la documentación de su arquitectura original orientada a objetos, la iAPX432. La intención de este icono es sugerir las fronteras de una abstracción, las líneas punteadas del contorno indican que generalmente los clientes operan sobre las instancias de una clase, no sobre la clase misma.

Una operación denota un servicio proporcionado por la clase y normalmente se les asigna un nombre y se escriben dentro del icono de clase, y se distinguen de los atributos porque se les añade un par de paréntesis o, si es necesario para el propósito del diagrama, se escribe la descripción completa de la operación:

- **N ()** Nombre de la operación solamente
- **R N (Argumentos)** Clase de retorno de la operación, nombre y parámetros formales (si los hay)

3.4.2 Relaciones de clases

Las clases no existen de manera aislada en un sistema sino que se relacionan con otras clases de diversas maneras, formando así lo que se llama una estructura de clases.

Las relaciones básicas entre clases son cuatro y son las siguientes: relaciones de asociación, de herencia, de posesión y de uso o utilización. Los iconos que se utilizan para su representación se pueden ver en la figura 3.4. Cada una de estas relaciones puede incluir una etiqueta textual, la cual documenta el nombre de la relación o sugiere su propósito.

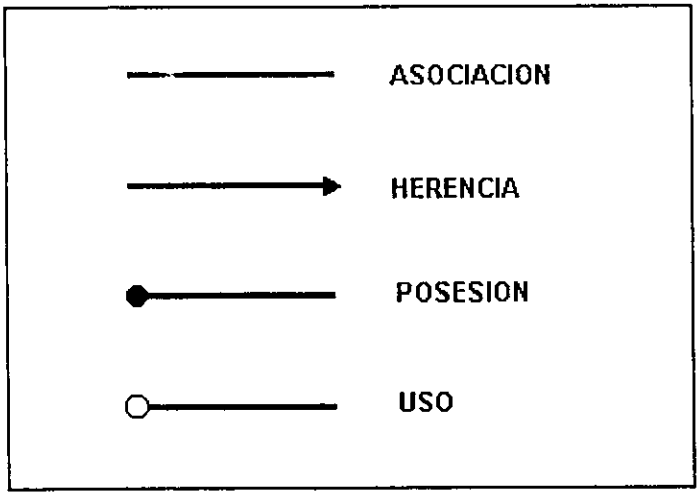


Figura 3.4 Iconos de relación entre clases

El icono de asociación conecta a dos clases, estas asociaciones pueden ser uno a uno, uno a muchos o muchos a muchos, es decir, una instancia de una clase puede asociarse con solo una instancia de otra clase, o una instancia puede asociarse con muchas instancias de otra clase.

Para ilustrar estas asociaciones con un ejemplo, consideremos un sistema automatizado para un punto de venta, dos de las **abstracciones clave** son productos y ventas, en la figura 3.5 se muestra una **asociación** simple entre estas dos clases: la clase **PRODUCTO** contiene los productos que se venden como parte de una venta, y la clase **VENTA** denota la operación por la cual varios productos acaban de venderse.

Una asociación uno a uno se daría en las operaciones de venta con tarjeta de crédito. A cada venta le corresponde una y solo una transacción de tarjeta de crédito, y a cada transacción le corresponde una sola venta.

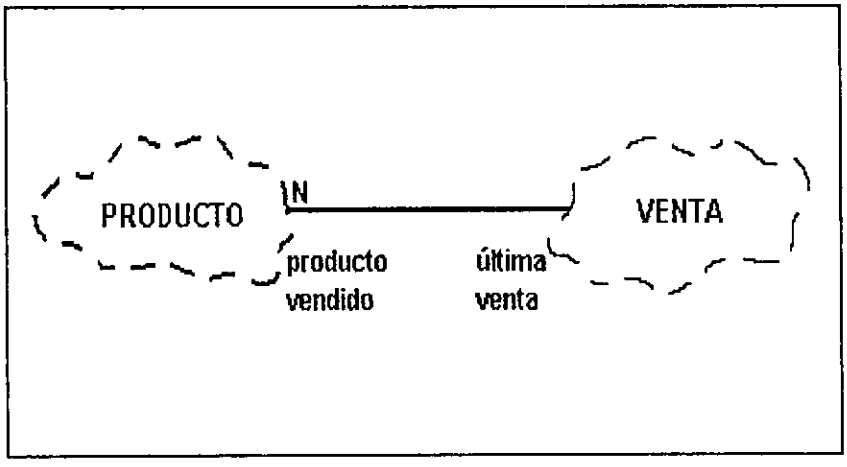


Figura 3.5 Asociación entre las clases VENTA y PRODUCTO

Una asociación uno a muchos sería cuando para cada instancia de la clase **VENTA** existe cero o más instancias de la clase **PRODUCTO**, y por cada producto existe exactamente una venta.

En una relación muchos a muchos, cada instancia de la clase **CLIENTES** podría realizar una transacción con una instancia de la clase **VENDEDOR**, y cada uno de esos vendedores podría interactuar con muchos clientes distintos.

Para denotar el número de enlaces entre cada instancia de la clase origen y las instancias de la clase destino, se utiliza una etiqueta de cardinalidad y se escribe en el extremo de destino de una asociación.

Los tres tipos de asociación restantes son una variación del icono de asociación, se puede decir que este último es el más general. Durante el desarrollo de una aplicación, las asociaciones tienden a evolucionar en el sentido de que primero se establece la existencia de una conexión semántica entre dos clases, y a medida que se van tomando desiciones tácticas acerca de la naturaleza de su interrelación, estas se van refinando ya sea como relación de herencia, posesión o de utilización.

El icono de herencia significa una relación de generalización/especialización y se escribe como una asociación con cabeza de flecha. La flecha apunta a la superclase y el extremo opuesto de la asociación designa a la subclase la cual hereda la estructura y el comportamiento de su superclase de acuerdo a las reglas del lenguaje de programación elegido para la implementación.

También una clase puede heredar de una o más superclases, en el primer caso se trata de herencia simple y en el segundo caso de herencia múltiple, como se puede apreciar en el ejemplo de la figura 3.6. Además las relaciones de herencia no llevan indicaciones de cardinalidad.

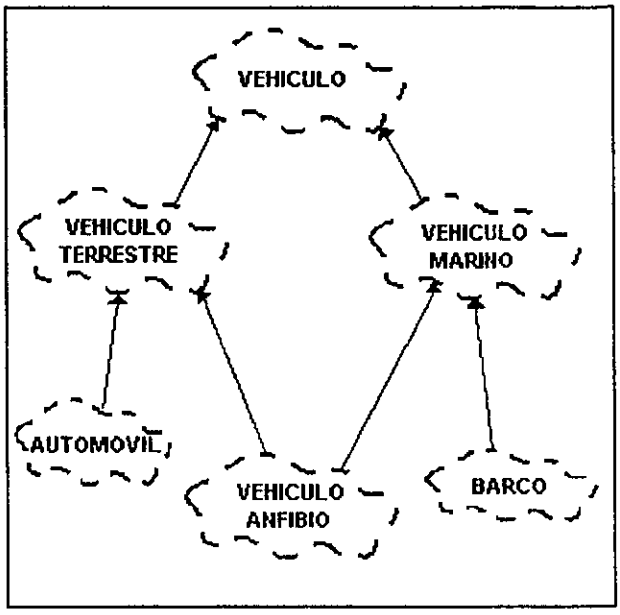


Figura 3.6 Herencia

El icono de posesión indica un relación todo/parte, también conocida como relación de agregación y se escribe como una asociación con un circulo relleno en el extremo que señala al agregado. La clase que esta en el otro extremo denota la parte cuyas instancias están contenidas en el objeto agregado.

El icono de utilización significa una relación cliente/servidor y se escribe como una asociación con un circulo vacío (opuesto al circulo del icono de posesión) en el extremo que señala al cliente. Esta relación indica que el cliente utiliza los recursos del servidor y depende del mismo porque le proporciona determinados servicios. Típicamente se utiliza para indicar la desición de que hay operaciones de la clase cliente que invocan operaciones de la clase servidora, o utilizan algún tipo de función cuyos argumentos son instancias de la clase servidora.

3.4.3 Categorías de clases

De la misma manera que una clase es un agregado de objetos que contiene operaciones e incluso otras clases, una categoría de clases es una colección lógica de clases, cada una de las cuales debe estar en una sola categoría de clases o en un nivel superior dentro del sistema. Al contrario que las clases, una categoría de clases no contribuye directamente al sistema con operaciones; lo hace indirectamente a través de las clases que contiene.

El icono para representar una categoría de clases se muestra en la figura 3.7, de la misma manera que una clase, se requiere un nombre para cada categoría de

clases. Se puede representar la arquitectura lógica de alto nivel de un sistema por medio de un diagrama de clases que contenga solo categorías de clases.

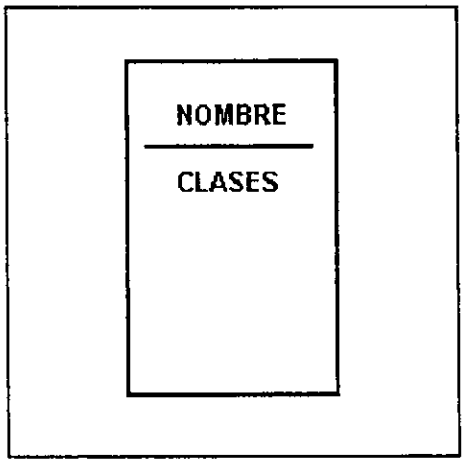


Figura 3.7 Icono de categoría de clases

En algunos lenguajes de programación no se tiene apoyo sintáctico para la implementación de las categorías de clases, sin embargo, en Smalltalk existen como una recomendación para organizar la biblioteca de clases. en C++ las categorías de clases están relacionadas con el concepto de "componentes".

3.5 Diagramas de Objetos

Un diagrama de objetos se utiliza para mostrar la existencia de objetos y sus relaciones en el diseño lógico de un sistema. Representa una vista de la estructura de objetos de un sistema.

Durante el análisis se usan para indicar la semántica de los escenarios principales, los cuales constituyen una guía que nos indica el comportamiento del sistema.

Durante el diseño se usan diagramas de objetos para ilustrar la semántica de los mecanismos en el diseño lógico del sistema.

Los dos elementos esenciales de un diagrama de este tipo son los objetos y sus relaciones.

3.5.1 Objetos

Un objeto se representa en un diagrama de objetos casi igual que una clase, solamente que el contorno del icono es continuo como el que se muestra en la figura 3.8. De la misma manera que se hace con los diagramas de clases, se puede dibujar una línea horizontal para dividir el texto del interior del icono en dos regiones, una que denota el nombre del objeto y otra que nos indica los atributos del objeto.

Puede escribirse solamente el nombre del objeto o el nombre y la clase a la que pertenece utilizando la sintaxis del lenguaje de programación elegido para la implementación.

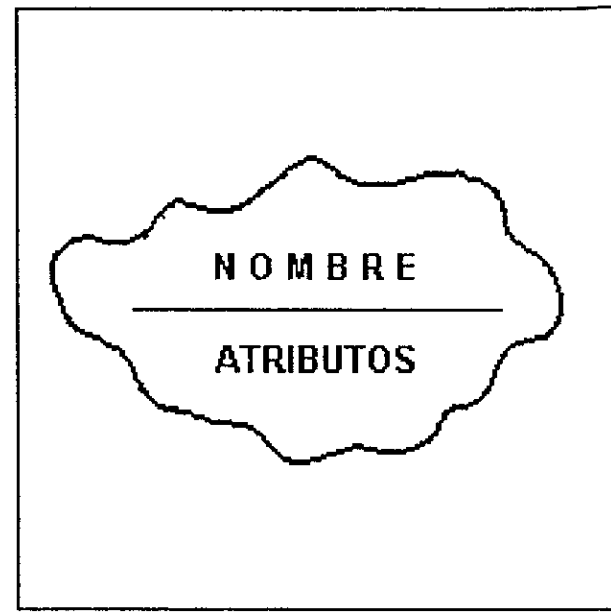


Figura 3.8 Icono para representar un objeto

Si no se especifica la clase de un objeto, entonces la clase del objeto se considera anónima y no puede haber comprobación semántica alguna de las operaciones ejecutadas sobre o por parte del objeto, ni sobre la relación del objeto con cualquier otro objeto del diagrama.

Respecto a los atributos del objeto se puede usar la sintaxis descrita para las clases y sus atributos, e incluso se puede especificar una expresión por default para cada atributo. Los nombres de estos deben hacer referencia a un atributo definido en la clase del objeto o en cualquiera de sus superclases.

En los diagramas de objetos también se pueden incluir iconos que denotan utilidades de clase y metaclases, ya que ambas denotan cosas parecidas a objetos, sobre los que se puede operar y que operan sobre otros objetos.

3.5.2 Relaciones entre objetos

Los objetos interaccionan a través de sus enlaces con otros objetos. Puede existir un enlace entre dos objetos, si y sólo si, existe una relación entre sus clases correspondientes. Esta relación establece una via de comunicación entre instancias de las clases y es la que posibilita que un objeto pueda enviar mensajes a otro.

En general, dado un objeto **A** con un enlace **L** hacia un objeto **B**, Se dice que **A** puede invocar cualquier operación aplicable a la clase de **B**, La inversa también es cierta para las operaciones invocadas por **B** sobre **A**. Cualquier objeto que invoque una operación sobre otro se le conoce como **cliente**; y cualquier objeto que suministre la operación se le conoce como el **proveedor** o **servidor**.

El icono de asociación entre objetos se muestra en la figura 3.9, además se pueden agregar una serie de mensajes. Cada mensaje consta de los siguientes tres elementos:

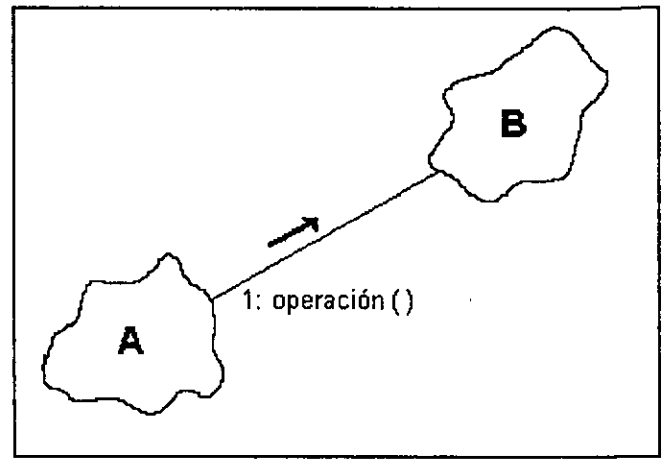


Figura 3.9 Muestra la relación entre los objetos A y B, por medio de un enlace; así como la dirección de la invocaión de una operación del objeto cliente (A) sobre el objeto servidor (B) y un número de secuencia (1) que indica el orden en que una operación se invoca.

---->	Un símbolo de sincronización el cual indica la dirección de la invocación
operación ()	Una invocación de operación
número (1,2...)	Opcionalmente, un número de secuencia

3.6 Diagramas de transición de estados

Un diagrama de transición de estados es un gráfico dirigido en el cual los nodos representan los estados del sistema y los arcos representan las transiciones entre dichos estados. Su utilidad estriba en que nos permiten especificar e implementar los aspectos de control de un sistema, también se utilizan para mostrar el espacio de estados de una clase determinada, los eventos que provocan el cambio de un estado a otro, y las acciones que resultan de ese cambio de estado.

Durante el análisis, se usan diagramas de transición de estados para indicar el comportamiento dinámico del sistema completo.

Durante el diseño, se usan para capturar el comportamiento dinámico de las clases individualmente o de un conjunto de ellas.

Los elementos esenciales de un diagrama de transición de estados son los estados y las transiciones entre estados.

3.6.1 Estados

Un estado es la abstracción de los valores de los atributos y las relaciones de un objeto. Un conjunto de valores están agrupados dentro de un estado de acuerdo con las propiedades que afectan el comportamiento del mismo. Por ejemplo, el estado de un banco puede ser cualquiera de los siguientes: SOLVENTE o INSOLVENTE, dependiendo, de si su capital excede sus pasivos u obligaciones a pagar.

Un estado corresponde a un intervalo entre dos eventos recibidos por un objeto. los eventos representan puntos en el tiempo; los estados representan intervalos de tiempo. El estado de un objeto depende de la secuencia anterior de eventos que ha recibido.

Un estado muchas veces se asocia con actividades contínuas que tienen un tiempo de duración, tal como el volar desde una ciudad a otra. Los eventos y los estados se complementan el uno al otro porque un evento separa dos estados y un estado separa dos eventos.

En la figura 3.10 se muestran los iconos para representar un estado específico, el cual recibe un evento de entrada y un evento de salida o respuesta. El nombre de un estado debe sugerir su propósito explícitamente.

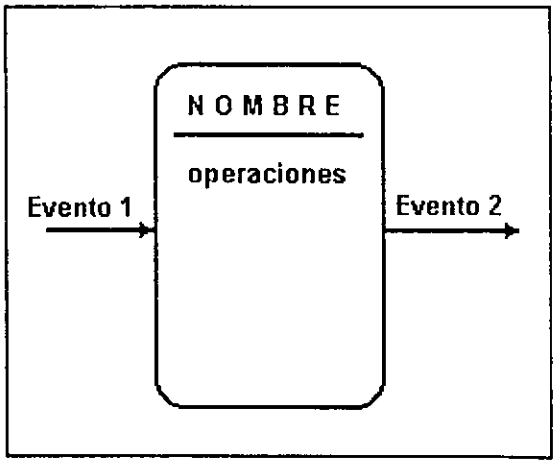


Figura 3.10 Iconos de estado y eventos

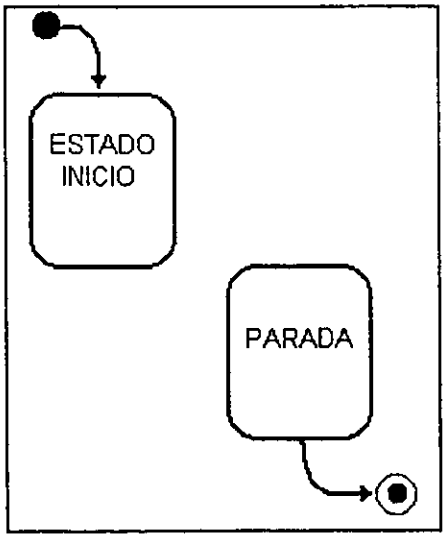


Figura 3.11 Iconos para designar un estado de inicio y un estado de parada respectivamente

En todo diagrama de estado debe haber exactamente un estado de partida, que se designa escribiendo una transición sin etiqueta al estado desde un icono especial que aparece como un circulo relleno. No necesariamente debe haber un estado de parada, pero, sí se necesita designarlo, se hace dibujando una transición sin etiqueta desde el estado hacia un icono especial que tiene forma de un icono relleno dentro de otro vacío ligeramente mayor. Ver figura 3.11

3.6.2 Transición entre estados

Un evento es algún suceso que puede causar un cambio de estado en un sistema. A este cambio de estado se le llama transición entre estados.

Como se mencionó anteriormente, un evento es cualquier cosa que sucede en un punto en el tiempo, no tiene duración, es solamente la ocurrencia de algo en un instante dado. Por ejemplo, cuando un usuario hace click en un botón del mouse, la ejecución de una transacción en un banco o la partida de un vuelo hacia una ciudad.

Un evento puede preceder o seguir a otro, o bien, dos eventos pueden estar no-relacionados, en este caso se dice que son concurrentes, porque la concurrencia ocurre cuando dos o más tareas, actividades o eventos, cuya ejecución puede traslaparse en el tiempo, sin embargo, no tienen efecto alguno uno sobre el otro, no importa si la localización física de los dos eventos es distante o no.

En el modelado de sistemas no se debe intentar establecer un orden entre eventos concurrentes por la simple razón de que pueden ocurrir en cualquier orden. Cualquier modelo realista de un sistema distribuido debería incluir eventos y actividades concurrentes.

Un evento conlleva información de un objeto a otro, es una forma de transmisión de información entre objetos, si así lo queremos ver; ésta información puede ser una simple señal de que algo ha ocurrido, o bien, pueden ser diversos valores de los datos, dichos valores constituyen los ATRIBUTOS de un evento. Los atributos se muestran entre paréntesis después del nombre de un evento, por ejemplo:

Salida de un aeroplano (línea aérea, número de vuelo, ciudad)
Oprimir botón del mouse (botón, localización)
Entrada de una cadena de caracteres (texto)
Dígitos marcados (dígito)

3.7 Diagramas de interacción

Un diagrama de interacción se usa para realizar un rastreo de la ejecución de un **escenario**. En realidad un diagrama de interacción es simplemente otra manera de representar un diagrama de objetos.

Un **escenario** es una secuencia de eventos que ocurren durante la ejecución de un sistema. El campo de acción de un escenario puede variar; puede incluir todos los eventos de un sistema, o sólo aquellos generados por ciertos objetos en el

sistema. Podemos ver un escenario como un registro histórico de la ejecución de un sistema o como la experimentación de la ejecución de un sistema propuesto.

A continuación, se muestra un ejemplo de un escenario para utilizarse en una línea telefónica. Solamente contiene eventos que afectan a la línea telefónica.

- **emisor descuelga auricular**
- **comienza tono de marcar**
- **emisor marca dígito (5)**
- **finaliza tono de marcar**
- **emisor marca dígito (9)**
- **emisor marca dígito (5)**
- **emisor marca dígito (4)**
- **emisor marca dígito (3)**
- **emisor marca dígito (6)**
- **emisor marca dígito (9)**
- **el teléfono solicitado (receptor) comienza a timbrar**
- **se escucha tono en el teléfono del emisor**
- **persona solicitada descuelga auricular**
- **el teléfono solicitado (receptor) deja de timbrar**
- **desaparece tono en el teléfono receptor**
- **los teléfonos son conectados**
- **persona solicitada cuelga auricular**
- **los teléfonos son desconectados**
- **emisor cuelga auricular**

Escenario para una llamada telefónica

En este ejemplo, cada evento transmite información desde un objeto a otro; cuando sucede el evento "**comienza tono de marcar**", se transmite una señal desde la línea telefónica hacia el emisor que hace que éste, pueda empezar a marcar dígitos.

El siguiente paso después de escribir un escenario es identificar los objetos emisor y receptor de cada evento. La secuencia de eventos y los objetos intercambiando eventos se pueden mostrar en un escenario aumentado llamado **diagrama de interacción**[5]. Es un diagrama que aparece en forma tabular y muestra los objetos de interés que se escriben horizontalmente a lo largo de la parte superior del diagrama, se dibuja una línea vertical discontinua bajo cada objeto y los eventos (que pueden ser mensajes o la invocación de operaciones) se denotan con una flecha horizontal que va del objeto emisor hacia el objeto receptor, tal como se muestra en la figura 3.12. El tiempo se incrementa de arriba hacia abajo, el espacio ocupado es irrelevante; sólo se muestra la secuencia de los eventos, no el tiempo exacto de ejecución.

[5] Este diagrama es una generalización de los diagramas de traza de eventos (events trace diagrams, en el original en inglés) de J. Rumbaugh y de los diagramas de interacción de Jacobson.

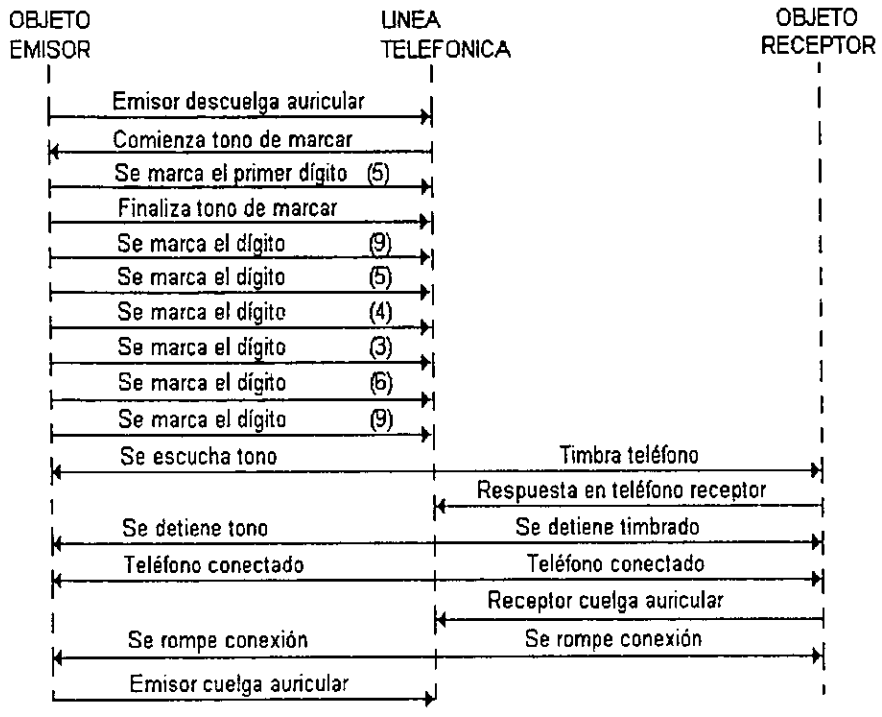


Figura 3.12 Diagrama de interacción para el ejemplo de la llamada telefónica

3.8 Diagramas de módulos

Un diagrama de módulos se usa para mostrar la asignación de clases y objetos a módulos en el diseño físico de un sistema. Con un solo diagrama de módulos podemos representar la arquitectura de módulos de un sistema o solo una parte de ella.

Un diagrama de módulos es un gráfico cuyos nodos representan módulos y los arcos representan las relaciones entre esos módulos.

3.8.1 Módulos

Un módulo es la unidad de código que sirve como bloque de construcción para la estructura física de un sistema; es un segmento de programa que contiene declaraciones, expresadas en el vocabulario de algún lenguaje de programación, dichas declaraciones conforman la implementación física de las clases y objetos del diseño lógico de un sistema.

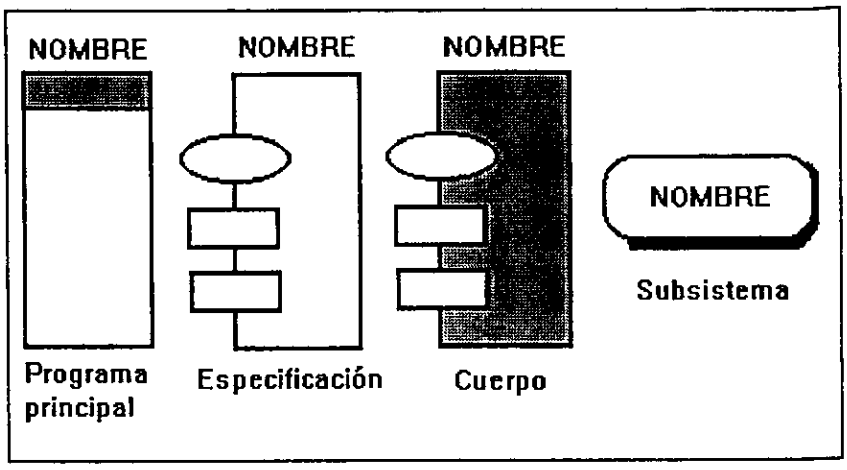


Figura 3.13 Iconos para representar módulos y un subsistema

En la figura 3.13 se muestran los iconos para representar varios tipos de módulos. Los tres primeros iconos denotan archivos que se distinguen entre si por su función, el cuarto denota un subsistema.. El icono de programa principal denota un archivo que contiene la raíz del programa. Por ejemplo, en C++ podría ser un archivo con extensión **.cpp** que contiene la definición de la función **main()**. El icono de especificación y el de cuerpo denotan archivos que contienen, respectivamente, la declaración y la definición de las entidades. En C++, por ejemplo, los módulos de especificación denotan archivos con la extensión **.h** y los módulos de cuerpo denotan archivos **.cpp**. El nombre de cada módulo es generalmente el nombre del archivo físico correspondiente.

3.8.2 Dependencias entre módulos

La única relación que puede darse entre dos módulos es una dependencia de compilación, que se representa por medio de una flecha que apunta al módulo respecto al cual existe la dependencia. En C++ por ejemplo, se indica una dependencia de compilación mediante la sentencia **#include**. De la misma manera en ADA, las dependencias de compilación se indican con cláusulas **whith**.

3.8.3 Subsistemas

Se utiliza el concepto de subsistemas principalmente porque los sistemas grandes se pueden descomponer en varios módulos, quizá cientos o miles, lo cual dificulta el poder comprender la arquitectura física de un tales sistemas si no se toma la estrategia de agrupar módulos relacionados en subsistemas.

Los subsistemas sirven para particionar el modelo físico de un sistema. La figura 3.13 muestra el icono para representar a un subsistema, al igual que los módulos, se requiere un nombre para cada subsistema.

Algunos de los módulos englobados por un subsistema pueden ser públicos, lo que significa que son exportados por el subsistema y por lo tanto utilizables fuera del mismo. Otros módulos, pueden ser parte de la implementación del subsistema y por lo tanto no son utilizables por ningún otro módulo fuera del subsistema. Por

convención, todo módulo de un subsistema se considera público, a menos que defina de otra forma.

Un subsistema puede tener dependencia con respecto a otros subsistemas, o módulos y un módulo puede tener dependencia respecto a un subsistema. Se aplica el mismo icono de dependencia utilizado hasta ahora (una flecha) para indicar tal dependencia.

3.9 Diagramas de procesos

Un diagrama de procesos se usa para mostrar la asignación de procesos a procesadores en el diseño físico de un sistema. Por medio de un diagrama de procesos podemos representar una vista de la arquitectura de procesos de un sistema pues estos nos indican que procesadores y dispositivos se utilizan como plataforma de ejecución del sistema.

Los elementos esenciales de un diagrama de procesos son: los procesadores, los dispositivos y sus conexiones.

3.9.1 Procesadores

Como bien sabemos, cuando hablamos de un procesador nos referimos al hardware capaz de ejecutar los programas. El icono que se utiliza en la notación para representar un procesador se indica en la figura 3.14. Se requiere asignar un nombre para cada procesador y no hay restricciones en cuanto a ellos se refiere porque solamente denotan entidades del hardware, más no del software. Se puede añadir al icono de procesador una lista de los procesos que se ejecutan en dicho procesador.

3.9.2 Dispositivos

Un dispositivo, es una parte del hardware que no tiene la capacidad para ejecutar programas. Ejemplos de dispositivos pueden ser terminales, modems, impresoras, scanners, sensores, etc. En la figura 3.14 se muestra el icono utilizado para representar a un dispositivo, el cual también requiere un nombre y no hay restricciones sobre los nombres de dispositivos.

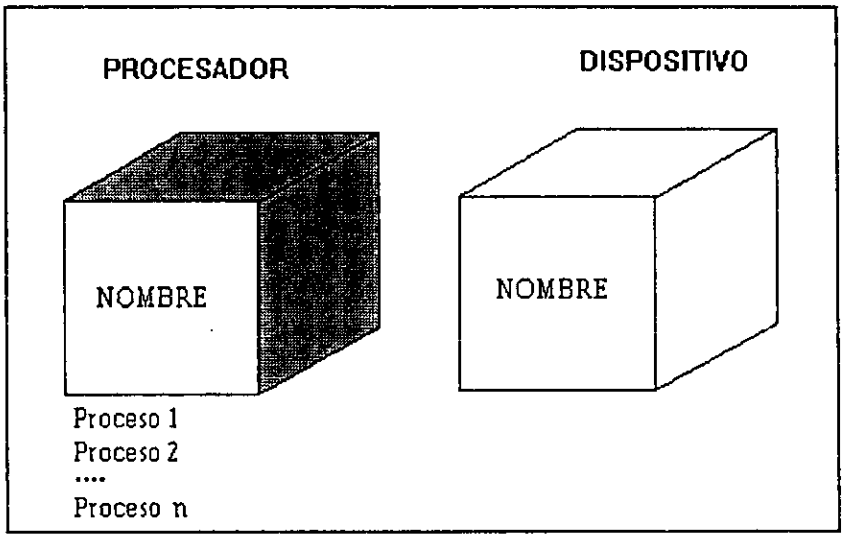


Figura 3.14 Iconos de procesador y dispositivo

3.9.3 Conexiones

Las conexiones representan la comunicación entre los procesadores y los dispositivos, esta comunicación se expresa mediante líneas no dirigidas, se puede indicar la conexión entre un procesador y un dispositivo, un procesador y otro procesador, o un dispositivo y otro dispositivo. Una conexión puede representar por ejemplo, una conexión Ethernet, o un cable RS232. También puede representar acoplamientos menos directos, como comunicaciones remotas entre procesadores.

La notación descrita hasta este punto es aplicable tanto a sistemas pequeños que sólo constan de unas cuantas clases, como a otros que constan de muchas más definiciones de clases.

La notación es aplicable a un desarrollo incremental e iterativo, pues los diagramas que se obtienen evolucionan durante el proceso de diseño a medida que se toman nuevas decisiones de diseño y se establece un mayor detalle.

Otra ventaja de esta notación es aplicable a muchos lenguajes de programación orientados a objetos.

A continuación se describe el proceso de desarrollo en el cual se aplica directamente la notación descrita hasta este punto.

3.10 El proceso de desarrollo

El desarrollo de cualquier proyecto de software no es algo trivial, sin embargo, muchos estudiantes de carreras relacionadas con la ingeniería de software, incluyendo los de matemáticas aplicadas y computación, buscamos algún método o herramienta sensacional cuya aplicación nos asegure que el proceso de desarrollo es fácil, ignorando muchas veces por completo la documentación de los sistemas, preocupándonos más por el aspecto que el software presenta ante el cliente que sobre el contenido del mismo. En otras palabras, buscamos "recetarios" para producir software y no pensamos que la aplicación rígida de una metodología puede conducirnos a obtener productos de software de baja calidad, que no satisfacen los requerimientos reales del cliente, sin embargo, en la práctica profesional es cuando nos damos cuenta de la realidad, y aceptamos la importancia que tiene crear ciertos documentos que servirán de respaldo de las decisiones tomadas en el diseño de un buen producto de software. En particular hago referencia a la importancia que tiene el análisis de sistemas, en este caso, el análisis de sistemas orientados a objetos, para respaldar las decisiones tomadas durante el diseño orientado a objetos y que nos conducirán a arquitecturas consistentes, sencillas y bien estructuradas. Estos dos últimos aspectos son la base principal del presente trabajo, y sobre los cuales gira todo el desarrollo de la tesina.

En los apartados siguientes se examina detalladamente el proceso incremental e iterativo del desarrollo de software orientado a objetos descrito por Booch, considerando el propósito, productos, y actividades de cada fase.

3.10.1 Consideraciones iniciales

Antes de explicar el proceso de desarrollo consideremos las propiedades de los proyectos de software exitosos y quizá las más importantes sean:

- El producto de software entregado satisface o rebasa las espectativas del cliente.
- El proyecto se desarrolla dentro de un marco económico aceptable y dentro de los tiempos estimados.
- El producto es resistente al cambio en los requerimientos futuros.

Estas propiedades son el resultado de dos características comunes en todos los sistemas orientados a objetos[6].

- Todos tienen un fuerte visión arquitectónica.
- Aplican un ciclo de vida del desarrollo bien dirigido, iterativo e incremental.

3.10.1.1 Visión Arquitectónica. Abarca la estructura lógica y física del sistema, es decir, su estructura de clases y objetos organizados en diferentes niveles. Como casi siempre sucede, la arquitectura de un sistema no le interesa al usuario final, sin embargo, es fundamental para los diseñadores del software para construir sistemas que sean comprensibles, modificables, mantenibles y que puedan probarse contra la realidad.

Es necesario distinguir entre decisiones arquitectónicas estratégicas y tácticas. Una **decisión estratégica** tiene implicaciones globales en el sistema, por ejemplo, la definición de los mecanismos para la detección de errores, la construcción de interfases de usuario, las políticas para el manejo de memoria y persistencia de objetos, los enfoques para la sincronización de procesos en aplicaciones de tiempo real[7]..., todas estas son decisiones arquitectónicas estratégicas. Por el contrario una **decisión táctica** sólo tiene implicaciones arquitectónicas a nivel local, e involucra normalmente los detalles de la interfase y la implantación de una abstracción, por ejemplo, el protocolo de una clase, elegir un algoritmo en particular para implantar un método.

Para tener una visión arquitectónica fuerte se debe buscar un equilibrio entre estos dos tipos de decisiones, se debe prestar atención y cuidado tanto al diseño de las clases individuales como al comportamiento que estas tendrán en conjunto para satisfacer el comportamiento deseado del sistema.

3.10.1.2 Ciclo de vida iterativo e incremental. El proceso de desarrollo orientado a objetos es iterativo en el sentido de que conlleva el mejoramiento sucesivo de una arquitectura orientada a objetos, es decir, toma la experiencia y resultados de cada versión anterior para la siguiente iteración de análisis y diseño. El proceso es incremental en el sentido de que cada pasada por el ciclo análisis/diseño/evolución, conduce a refinar gradualmente las decisiones estratégicas y tácticas, conduciendo a su vez hacia una solución no solamente factible sino real, que refleja los requerimientos reales del usuario final.

---

[6] Booch, Grady. *Análisis y diseño orientado a objetos con aplicaciones*, segunda edición, Addison Wesley/Díaz de Santos, Wilmington, Delaware, E.U.A. 1996. pp. 263.

[7] *ibidem*, p.265

El ciclo de vida orientado a objetos es la antítesis del punto de vista del ciclo de vida tradicional de cascada (ver apartado 1.6.2 del capítulo I), no es un proceso estrictamente ni ascendente ni descendente, sino más bien es parecido al modelo espiral propuesto por Barry Boehm.[8]

El modelo propuesto por Boehm para el desarrollo de software, esta representado por la espiral como la que se muestra en la figura 3.15, define 4 actividades principales representadas por los 4 cuadrantes de la figura:

1. Planeación.- determinación de objetivos, alternativas y restricciones
2. Análisis de riesgo.- análisis de alternativas e identificación de resolución de riesgos
3. Ingeniería.- desarrollo del "siguiente nivel" del producto
4. Evaluación del cliente.- aceptación de los resultados de ingeniería

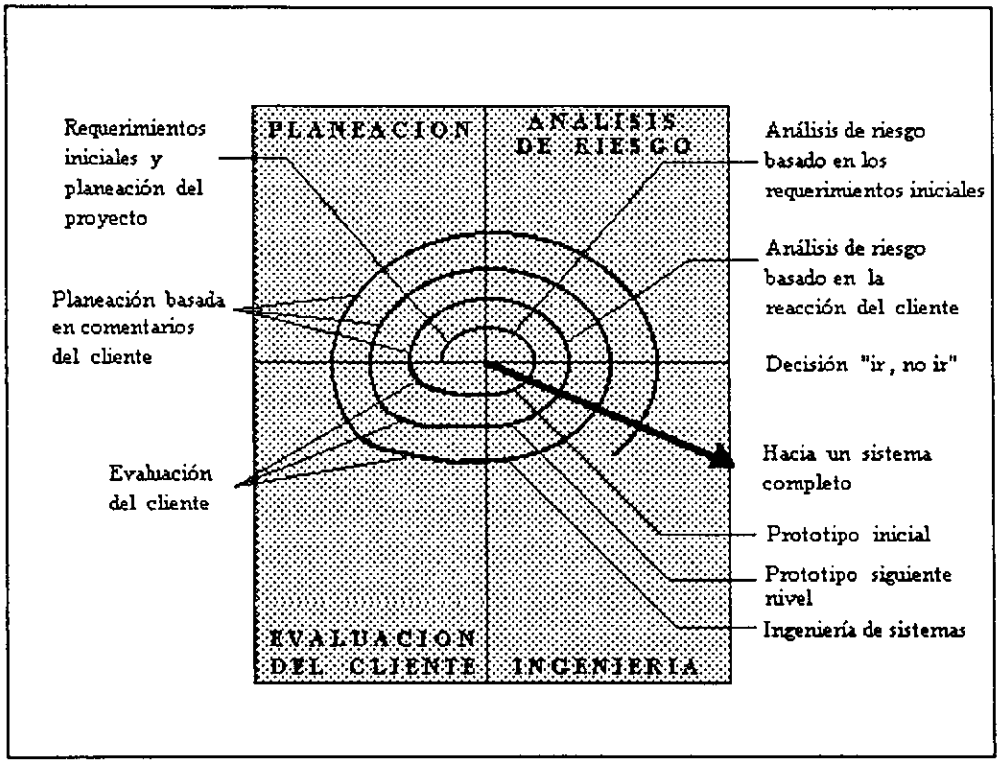


Figura 3.15 Modelo espiral para el desarrollo de software propuesto por B. Boehm.

Con cada iteración alrededor de la espiral (comenzando del centro), se construyen progresivamente versiones más completas del software.

Durante el primer circuito alrededor de la espiral, se definen los objetivos, alternativas así como las restricciones y los riesgos son identificados y analizados.

---

[8] Citado por Pressman, Roger S. *Software Engineering, a practitioner approach*, tercera ed. McGraw Hill, U.S.A. 1992. pp. 29-30.

Si el análisis de riesgo indica que hay incertidumbre en los requerimientos, entonces se puede utilizar el prototipo en el cuadrante de ingeniería para asistir al desarrollador y al cliente con el fin de definir el problema y refinar los requerimientos. El cliente evalúa el trabajo de ingeniería y hace sugerencias y posibles modificaciones. La siguiente fase de planeación y análisis de riesgo se basa en las sugerencias del cliente . En cada ciclo de alrededor de la espiral, la culminación del análisis de riesgo resulta en una desición del tipo "ir, no ir". Si el riesgo es muy grande, el proyecto puede terminarse.

Este modelo de desarrollo espiral es un enfoque muy realista para el desarrollo de sistemas de gran escala y es la base del proceso del diseño orientado a objetos, ya que al igual que éste utiliza un enfoque incremental e iterativo.

Booch, plantea que una organización de desarrollo de software puede asegurar un proceso maduro, cuando se reconcilia la necesidad de creatividad e innovación con el requerimiento de prácticas de control y medición del software y hace una división de los elementos del proceso de desarrollo en micro y macroelementos y los llama a su vez **microproceso y macroproceso de desarrollo,** el primero esta estrechamente relacionado con el modelo espiral de desarrollo de Boehm, es decir, tiene un enfoque iterativo e incremental y representa básicamente las actividades diarias del desarrollador(es). El macroproceso es utilizado para controlar al microproceso, y dirigir las actividades del equipo completo de desarrollo. En las siguientes secciones se explican las actividades que se llevan a cabo durante el proceso.

3.11 El microproceso de desarrollo

En la figura 3.16 se ilustra el microproceso de desarrollo planteado por Booch, cabe mencionar que la mayoría de los métodos de análisis orientados a objetos siguen un modelo parecido al que se presenta aquí y consiste básicamente en las siguientes cuatro actividades:

- Identificar las clases y objetos relevantes para dominio del problema
- Identificar la semántica de clases y objetos
- Identificar relaciones entre clases y objetos
- Especificar interfases e implementación de clases y objetos

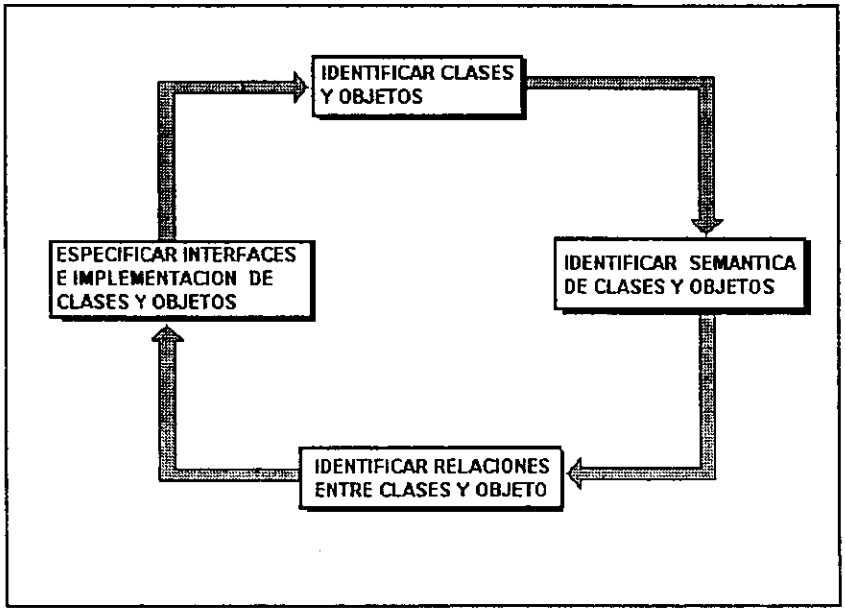


Figura 3.16 El microproceso de desarrollo

3.11.1 Identificación de clases y objetos.

Propósito. Forma parte del análisis y su propósito es identificar los objetos y las clases que forman parte del dominio del problema, definiendo así los limites del problema que se maneja, decidiendo lo que es importante para la aplicación y lo que no lo es. Como parte del diseño, esta actividad implica inventar nuevas abstracciones[9] que forman parte de la solución del problema y que se irán refinando conforme se avance en la implementación de las mismas, con la finalidad de crear abstracciones de nivel inferior que se puedan usar para construir otras de nivel superior, y para descubrir aspectos comunes entre las abstracciones existentes, lo cual conducirá a simplificar la arquitectura del sistema.

Productos. El producto principal de esta actividad es un diccionario de datos[10], el cual servirá como un depósito central para las abstracciones relevantes del sistema. Al principio algunas cosas podrían pasar a ser clases, otras objetos otras simplemente atributos de las abstracciones, conforme se avanza en la implementación, este diccionario se refinará añadiendo nuevas abstracciones

---

9 Recuérdese que una abstracción, en el contexto de la orientación a objetos es el proceso de centrarse en las características esenciales de un objeto; las cuales lo distinguen de todos los demás tipos de objetos.

10 Un diccionario de datos en el desarrollo orientado a objetos contiene la especificación de cada elemento de la arquitectura.

eliminando las que sean irrelevantes y consolidando aquellas con características similares en una sola.

Entre los beneficios que se derivan de la creación de un diccionario de datos tenemos los siguientes:

- El mantenimiento de un diccionario de datos ayuda a establecer un vocabulario común que puede utilizarse a lo largo de un proyecto de software.

- Puede ser un medio eficiente para hacer una rápida revisión de todos los elementos que conforman la arquitectura de un sistema. Esta característica es útil cuando hay nuevos miembros en el equipo de desarrollo, que tienen que orientarse rápidamente con el fin de integrarse a la solución que ya se está desarrollando.

Actividades. Es importante no confundirse y aclarar que cuando se está hablando de la identificación de las clases y objetos relevantes para una aplicación y del descubrimiento de las abstracciones clave, estamos hablando de lo mismo. De esta manera las principales actividades son simplemente el descubrimiento de las abstracciones clave y la creación de los mecanismos que satisfacen los requerimientos del problema.

¿Cómo se realizan estas actividades? En ambos casos se aplican cualquiera de los diferentes enfoques de análisis descritos en la sección 2.9. A continuación se explica una posible forma de hacerlo.

- Aplicar el enfoque clásico del análisis orientado a objetos (ver sección 2.9.1) para crear un conjunto de clases y objetos candidatos. Al inicio del ciclo de vida, las cosas tangibles y los roles que desempeñan es una buena fuente para la creación de estos candidatos.

- Aplicar la técnica de análisis de comportamiento (ver sección 2.9.2) para formar grupos de objetos que siguen un patrón de comportamiento similar.

- Utilización de fichas CRC (Clases/Responsabilidades/Colaboradores), ya que el uso de estas fichas constituyen una herramienta sencilla pero efectiva para analizar escenarios (Ver sección 2.9.4). Al principio del ciclo de vida se construyen escenarios muy generales que describen comportamientos amplios del sistema; en la medida que se avanza en el desarrollo, estos escenarios se tornan más detallados.

Es importante destacar que algunos de los objetos y clases que se identifican al principio del ciclo de vida orientado a objetos, seguramente serán equivocados, sin embargo, esto no debe preocuparnos, pues en la medida en que se va

aprendiendo más sobre el problema será posible reasignar responsabilidades clasificando objetos similares y otras veces dividiendo clases mas generales en subclases que cooperan para satisfacer un comportamiento deseado. Formando de esta manera algunos de los mecanismos de la solución.

¿Cuándo se completa la fase del microproceso? Esta fase se completa (más no se termina) cuando se tiene un diccionario de datos razonablemente estable, capaz de darnos una buena descripción de los elementos que conforman la arquitectura del sistema. Este diccionario no se termina sino, hasta fases muy avanzadas del proceso de desarrollo.

Medidas de Control La principal medida de control es que el diccionario no sufra cambios drásticos cada vez que se itera a través del microproceso. Un diccionario que cambia drásticamente puede ser signo de dos cosas: El equipo de desarrollo, aún no ha entendido el problema, o bien, la arquitectura es defectuosa en algún aspecto.

3.11.2. Identificación de la semántica de las clases y objetos

El propósito de la semántica de las clases y objetos es establecer el comportamiento y los atributos de cada abstracción que se identifica en la fase anterior. ¿Cómo se hace? Produciendo los diagramas de objetos y diagramas de interacción, porque por medio de estos se puede definir la semántica de los escenarios mediante los cuales se describen las funciones y responsabilidades[11] asignadas a los objetos indicándonos así los comportamientos del sistema.

Así mismo, como resultado de la asignación de responsabilidades a las diferentes abstracciones, se obtiene una actualización del diccionario de datos.

3.11.3 Identificación de las relaciones entre clases y objetos

Durante esta fase se identifican las relaciones entre clases y objetos, estableciendo los tipos de asociaciones que existen entre ellos, incluyendo las relaciones de herencia y agregación.

Los principales productos obtenidos, son los diagramas de clases y los diagramas de módulos. Estos diagramas permiten tener una visión más amplia de la arquitectura del sistema. Mediante los diagramas de clases, se establecen las asociaciones entre ellas durante el análisis.

---

[11] Una responsabilidad denota la obligación de un objeto para proporcionar cierto comportamiento

Durante el diseño se refinan estos diagramas mostrando las decisiones tácticas que se han tomado sobre sus relaciones de herencia, agregación, instanciación y uso.

3.11.4 Implementación de clases y objetos

Durante el análisis, el propósito de la implementación de clases y objetos es refinar las abstracciones existentes lo suficiente para descubrir nuevas clases y objetos a un mayor nivel de abstracción, las cuales se introducen en la siguiente iteración del microproceso.

Durante el diseño, el propósito de esta actividad es la representación de estas abstracciones, con miras la refinamiento sucesivo de las versiones ejecutables en el macroproceso.

Se producen las primeras versiones de código ejecutable, se puede entregar también diagramas de módulos que se utilizan para visualizar la correspondencia entre la arquitectura y la realización en forma de código.

Con el avance del desarrollo, se pueden utilizar herramientas CASE que generan código automáticamente partiendo de los diagramas, proceso que se conoce como ingeniería directa, o bien realizan ingeniería inversa, generando los diagramas a partir de la implementación.

En esta etapa, también se actualiza el diccionario de datos incluyendo las nuevas clases y objetos creados durante la implementación de las ya existentes.

Durante el análisis, esta fase se completa cuando se han identificado todas las abstracciones interesantes necesarias para satisfacer las responsabilidades de las abstracciones de nivel superior identificadas en esta pasada del microproceso.

Durante el diseño, esta fase se completa cuando se tiene un modelo ejecutable de las abstracciones.

3.12 El macroproceso de desarrollo

Este proceso es más amplio que el microproceso en el sentido de que los productos y actividades obtenidos permiten al equipo de desarrollo evaluar los posibles riesgos y realizar correcciones al microproceso.

Muchos de los elementos del macroproceso son buenas prácticas de administración del software, y por lo tanto se pueden aplicar tanto a sistemas orientados a objetos como a los no orientados a objetos.

El macroproceso descrito por Booch esta dirigido principalmente a la dirección técnica del equipo de desarrollo, cuya preocupación principal es un poco diferente de la del desarrollador individual. El interés común es que el software creado sea de alta calidad y que satisfaga las necesidades del cliente.

El cliente, por lo general no se interesa en como están implementadas las clases o si están organizadas en categorías de clases, ni en los detalles de los mecanismos creados; sino que se interesan más en las fechas, calidad, y completud. Por esta razón el macroproceso se centra en el riesgo y la visión arquitectónica, pues estos dos elementos tienen mayor impacto en los intereses del cliente.

Como se puede observar en la figura 3.17 el macroproceso propuesto por Booch atraviesa por las siguientes actividades.

- Desarrollar un modelo del comportamiento deseado del sistema (análisis)
- Crear una arquitectura para la implementación (diseño)
- Transformar la implementación mediante refinamiento sucesivo (evolución)
- Administrar la evolución postventa o postentrega (mantenimiento)

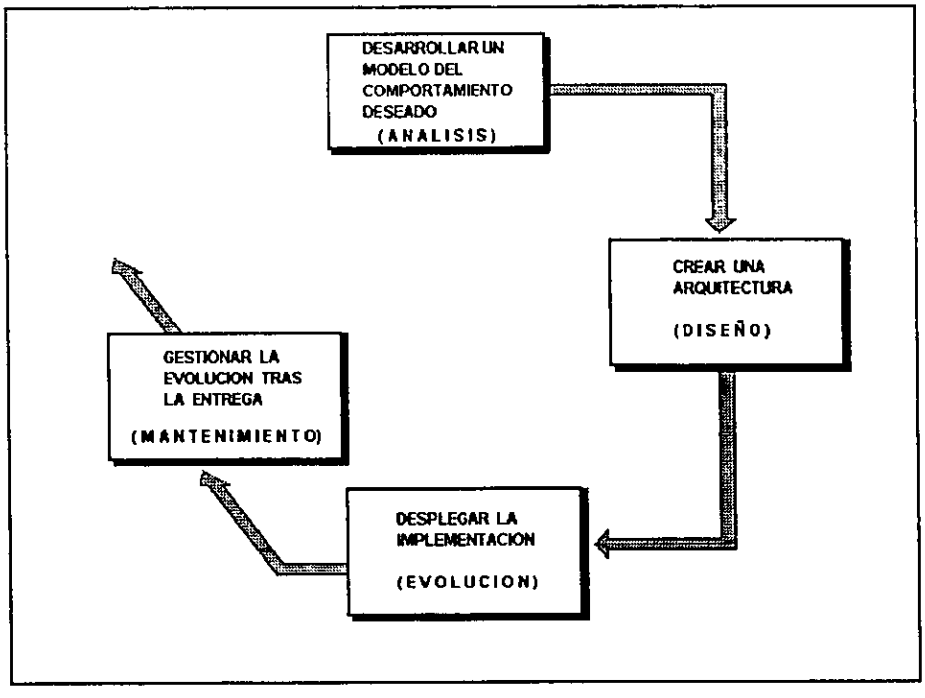


Figura 3.17 El macroproceso de desarrollo

3.12.1 Análisis. El análisis orientado a objetos, es un estado en el ciclo de vida del desarrollo de sistemas de software en el cual un problema del mundo real es examinado para entender sus requerimientos desde una perspectiva de clases y objetos que se encuentran en el vocabulario del dominio del problema. Este examen se hace sin planear todavía su implementación.

El propósito del análisis es proporcionar un modelo del comportamiento del sistema.

- El análisis se centra en el comportamiento no en la forma.
- Debe declarar lo que hace el sistema, no como lo hace.

Al centrarse en el comportamiento, se debe llegar a identificar los puntos funcionales del sistema, estos denotan los comportamientos observables exteriormente. Los puntos funcionales representan las transformaciones que el sistema realiza con su entorno, también pueden representar una medida de complejidad; cuanto mayor sea el número de puntos funcionales más complejo es el sistema. Se usan escenarios principales para ilustrar los comportamientos clave, y escenarios secundarios para mostrar algún comportamiento que se da bajo condiciones especiales. Para realizar narraciones de sucesos en los escenarios se usan técnicas de fichas CRC, y se usan los diagramas de objetos para ilustrar la semántica de cada escenario. Tales diagramas muestran los objetos que colaboran para conseguir la función. Además de los diagramas de objetos, se incluyen diagramas de clases.

Estos diagramas se reúnen en un documento formal de análisis de requisitos, que establece los requisitos del comportamiento del sistema tal como lo ilustran los diagramas.

Hay dos actividades principales asociadas al análisis: análisis del dominio y planificación de escenarios. El análisis de dominios busca identificar las clases y objetos que son comunes a un dominio de problema particular. Antes de implantar un sistema nuevo, es una buena medida estudiar otros existentes. De esta manera se obtienen beneficios de la experiencia de otros proyectos. En el mejor de los casos, como resultado de un análisis de dominios se puede descubrir que no se necesita desarrollar ningún software nuevo, sino que se pueden reutilizar o adaptar marcos de referencia existentes.

La otra actividad, la planificación de escenarios es la actividad central del análisis. Se llevan a cabo varios eventos en esta actividad:

- Identificar los puntos funcionales principales del sistema, reunirlos en grupos de comportamientos relacionados funcionalmente, considerar la agrupación de acuerdo con jerarquías de funciones.

- Para cada grupo de puntos funcionales, realizar una narración de eventos de un escenario, usando análisis de comportamiento, usar fichas CRC para discutir sobre cada escenario, cuando se va aclarando la semántica de cada escenario documentar con diagramas de objetos (que indican los objetos que contribuyen al comportamiento).

- Si es necesario, generar escenarios secundarios, que ilustren ciertos comportamientos bajo condiciones especiales.

- Actualizar el diccionario de datos, incluyendo las nuevas clases y objetos identificados para cada escenario, junto con sus roles y responsabilidades.

3.12.2 Diseño. El propósito del diseño es crear una arquitectura para la implementación del sistema, la arquitectura de un sistema orientado a objetos abarca su estructura de clases y objetos de tal manera que podemos usar sus respectivos diagramas para mostrar esta organización. A nivel arquitectónico es más útil mostrar el agrupamiento de clases en categorías de clases y el agrupamiento de módulos en subsistemas. Estos diagramas pueden entregarse como parte de un documento de arquitectura formal el cual debe revisar el equipo completo y debe actualizarse a medida que evoluciona la arquitectura.

Como podemos apreciar hasta este punto, los propósitos del análisis y del diseño son bastante diferentes. En el análisis buscamos modelar el mundo identificando las clases y objetos que conforman el vocabulario del dominio del problema; en el diseño se crean los mecanismos que proporcionan el comportamiento que éste modelo requiere. En esta fase trabajan conjuntamente usuarios y desarrolladores de un sistema, los une un vocabulario común derivado del dominio del problema.

¿Cuándo se debe iniciar el proceso de diseño? realmente el diseño puede comenzar tan pronto como se tenga un modelo razonablemente completo del comportamiento del sistema. No debemos caer en los extremos de hacer diseños prematuros en los que el desarrollo comienza antes de que se complete el análisis, ya que seguramente no conocemos ni sabemos lo suficiente acerca del problema para hacer un buen diseño. Ni de empezar a diseñar demasiado tarde porque se corre el riesgo de desperdiciar recursos al tratar de hacer un modelo de análisis muy detallado o perfecto y por lo tanto inalcanzable.

¿Cuándo se debe detener el proceso de diseño? Esta es una pregunta difícil pues como se dijo anteriormente se corre el riesgo de empezar el diseño demasiado pronto o demasiado tarde, en la práctica esta desición se toma sobre la marcha, de tal manera que se puede diseñar solamente las abstracciones clave y los mecanismos importantes de tal forma que se tenga un borrador que sea suficiente para la implementación, y aplazar a una fase posterior aquellos

aspectos que aportan poco o no aportan nada sobre el comportamiento observable del sistema. Hay que recordar que el ciclo de vida orientado a objetos es un proceso iterativo y creciente y por lo mismo las abstracciones clave de la aplicación se pueden ir refinando progresivamente. Booch señala la "regla del pulgar" que consiste en que uno puede detener el diseño cuándo las abstracciones clave no requieren más descomposición y que estas pueden ser compuestas desde componentes de software reutilizables.

3.12.3 Evolución. El propósito de la fase evolutiva es aumentar y cambiar la implantación mediante un refinamiento sucesivo, lo que conduce finalmente al sistema en producción. La evolución de una arquitectura es en gran parte cuestión de tratar de satisfacer una serie de restricciones que compiten entre si, como pueden ser la funcionalidad, el tiempo y el espacio.

El producto principal de la evolución es una serie de versiones ejecutables que representan sucesivos mejoramientos de la versión inicial de la arquitectura. Los productos secundarios incluyen prototipos de comportamiento. Un prototipo de comportamiento sirve para explorar algún elemento aislado del sistema, tal como un nuevo algoritmo o un modelo de alguna interfase de usuario o un esquema de base de datos. Su propósito es explorar rápidamente alternativas de diseño, de tal manera que se puedan resolver pronto áreas de riesgo sin poner en peligro las versiones de producción.

Booch, especifica que en la práctica se esperan los siguientes tipos de cambios durante la evolución de un sistema:

* Añadir una nueva clase
* Cambiar la implementación de una clase
* Cambiar la representación de una clase
* Reorganizar la estructura de clases
* Cambiar la interfase de una clase

Cualquiera de estos cambios se producen por diferentes razones, y cada uno implica un costo distinto. Por ejemplo se pueden añadir nuevas clases cuando se descubren nuevas abstracciones clave o se crean nuevos mecanismos, tales cambios no suelen tener consecuencias en términos de recursos de computación.

Cuando se añade una nueva clase, se debe ver en que lugar encaja de la estructura de clases existente. El cambio en la implementación de una clase, por lo general no resulta costoso. Sin embargo, los otros tres tipos de cambio conllevan un mayor costo porque implican la recompilación de código.

Por ejemplo en el caso de cambiar la representación de una superclase afecta a la representación de todas sus subclases: hay que recompilar su interfase, su implementación y todos sus clientes (es decir, sus subclases e instancias), todos

los clientes de sus clientes y así sucesivamente. La reorganización de una estructura de clases de un sistema es más frecuente, indica Booch, y por lo general consiste en cambios en las relaciones de herencia, añadir nuevas clases y trasladar responsabilidades e implementación de métodos comunes a clases más altas dentro de la estructura de clases. Booch indica, que en la práctica la reorganización de la estructura de clases es frecuente al principio, y después se estabiliza con el tiempo en la medida que los desarrolladores van comprendiendo mejor como trabajan juntas todas las abstracciones clave.

Para los sistemas pequeños el costo de la recompilación no representa mayor problema porque recompilar un programa completo puede tardar unos cuantos minutos, en cambio, para grandes sistemas, la cosa es muy distinta, ya que el recompilar un programa de cientos de miles de líneas, podría, por decir algo llevar incluso medio día o más de tiempo de computador.

Se puede decir que esta fase se completa con éxito cuando la calidad y la funcionalidad de las versiones son suficientes para expedir el producto.

3.12.4 Mantenimiento. Es la actividad de administrar la evolución postventa. Esta fase es una continuación de la fase anterior, excepto en que la modificación de la arquitectura del sistema es menos preocupante. En vez de eso, se realizan cambios al sistema en la medida que se agregan requerimientos y se eliminan errores. Los requerimientos de un sistema cambian en el transcurso del tiempo, lo cual nos hace diferenciar entre en la conservación de un sistema y su mantenimiento. Durante el mantenimiento, se les pide a los desarrolladores diferentes mejoras al sistema existente; pero sucede que muchas veces estos desarrolladores no son los mismos que diseñaron el sistema, sino un grupo diferente de personas. La conservación por el contrario conlleva el uso de recursos de desarrollo excesivos para mantener un sistema que envejece y que frecuentemente tiene una arquitectura diseñada de manera deficiente y es, por lo tanto difícil de comprender y de modificar. Esto implica tomar una desición de tipo económico: si el costo de mantener este software es mayor que el costo de desarrollar un sistema nuevo, lo más razonable es "enterrar" el sistema viejo y desarrollar uno nuevo.

Ya que el mantenimiento es en cierto sentido la evolución continuada de un sistema, sus productos son parecidos a los de la fase anterior. Además, el mantenimiento implica administrar una lista de nuevas tareas. Por lo general, inmediatamente después de que un sistema se encuentra en una fase operativa o llámese en producción, los usuarios finales ya tienen una serie de mejoras o modificaciones que les gustaría se realizaran en versiones posteriores y que posiblemente no se efectuaron en el producto inicial por razones de tipo económico. Una lista de nuevas tareas sirve como medio para recoger errores y requisitos de mejoramiento, de tal manera que se les pueda dar prioridad para futuras versiones.

CAPÍTULO IV

TENDENCIAS FUTURAS DE LAS APLICACIONES ORIENTADAS A OBJETOS

4.1 Objetivos

Los objetivos de este capítulo son:

Abordar el diseño de un sistema complejo real, del área de manejo de recursos forestales en México.

Describir las principales tendencias del software posibilitadas por las ventajas de la adopción de la programación orientada a objetos.

Introducción

La década de los años 80 se caracteriza por el desarrollo de aplicaciones posibilitadas por los lenguajes procedimentales tradicionales, entre las principales aplicaciones se tiene a hojas de cálculo, bases de datos, procesadores de texto, diseño asistido por computadora (CAD), entre otras.

En la década de los años 90, creció la necesidad de desarrollar y mantener sistemas de software complejos en ambientes competitivos y dinámicos, y que además soporten dominios de aplicación tales como la inteligencia artificial para desarrollar sistemas expertos que se aplican en diferentes áreas de la industria tal como manufactura, transportación, capacitación y servicios financieros; también tenemos aplicaciones multimedia que integran el manejo de información gráfica, de voz y texto; el desarrollo de lenguajes visuales; aplicaciones de simulación y en general aplicaciones de la ciencia y la ingeniería.

Todas estas aplicaciones requieren la utilización de la tecnología orientada a objetos, la cual como hemos visto en los capítulos anteriores ha cambiado nuestra perspectiva del desarrollo de sistemas. Particularmente, la programación orientada a objetos nos ha permitido pasar de una colección de funciones o subrutinas a construir sistemas basados en la combinación de componentes de software, que enfatizan la creación de *bibliotecas de clases* y *marcos estructurales* para dominios específicos.

Esto posibilita la construcción de sistemas complejos cuya tendencia es hacia la integración de sistemas de diferentes áreas de aplicación y hacia la reutilización de bibliotecas de clases-y marcos estructurales.

En este capítulo, primeramente se abordará el ambiente de aplicaciones agropecuarias con el fin de ejemplificar el proceso de desarrollo descrito en el capitulo III, posteriormente se abordarán brevemente algunas aplicaciones del área de la inteligencia artificial.

4.2 Aplicaciones actuales, orientadas a objetos en el área agropecuaria

Entre las aplicaciones que requieren la aplicación de la tecnología orientada a objetos se cuenta a los sistemas de apoyo a la investigación científica, los cuales actualmente les interesan a los principales centros de investigación en cómputo en el mundo.

Particularmente, trataré el ambiente de aplicaciones agropecuarias, ya que es donde pude obtener más información debido a mi trabajo en el *Colegio de Postgraduados en Ciencias Agrícolas*, donde hay gran cantidad de ejemplos de sistemas cuya complejidad demanda de las técnicas de análisis, diseño y desarrollo orientados a objetos. Un ejemplo lo constituye el proyecto **SAICA** (*Sistema de Apoyo a la Investigación Científica Agropecuaria*). Otros ejemplos se encuentran en el manejo de recursos naturales, estudios de suelos, de propagación de plagas y otras áreas más.

En los sistemas de planeación y manejo de los recursos naturales, la toma de decisiones sobre el uso adecuado de dichos recursos descansa en la utilización de grandes volúmenes de datos que se obtienen de diferentes fuentes tales como: inventarios de campo, encuestas, fotografías aéreas y mapas; el análisis de estos datos requiere del uso de sistemas de información geográfica (GIS[1]), paquetes estadísticos, sistemas de manejo de bases de datos (DBMS[2]) entre otros. Sin embargo las funciones que proporcionan estos sistemas, no se encuentran integradas dentro de un solo ambiente; es decir, que su utilización para resolver un problema es usarlas separadamente cada una de ellas.

Actualmente, se está desarrollando en el Colegio de Postgraduados un sistema para la planeación del manejo integral de los recursos forestales, para su desarrollo se esta empleando un modelo orientado a objetos llamado **REDAC** (**RE**presentación de **DA**tos y **C**onocimientos), sin embargo, obtener la información sobre este modelo, implicaba esperar algunos meses, por lo que decidí tomar el sistema para aplicar la metodología de Booch, únicamente para que sirva como ejemplo de un caso práctico y que además responde a las necesidades de México y no tomar ejemplos que reflejan la realidad de otro país. A continuación se da una breve introducción al sistema y se da una descripción general para diseñar una solución al mismo.

---

[1] Geographical Information Systems

[2] Data Base Manegement Systems

4.3 Sistema para la planeación del manejo integral forestal (SIPLAMIF)

En este caso prácticó, se pretende diseñar un sistema para la planeación de recursos forestales, aplicando el proceso de análisis y diseño de sistemas con orientación a objetos explicado en el capítulo anterior.

4.3.1 Antecedentes

A partir de 1986 con la expedición de la nueva Ley Forestal[3], surge el concepto de **Manejo Integral Forestal** (MIF), complementado por la también nueva Ley General de Equilibrio Ecológico y Protección al Ambiente, ambas leyes establecen un marco legal al sector forestal y en especial a los aprovechamientos forestales. La realización de dichos aprovechamientos requieren de los estudios del MIF.

El concepto del MIF debe entenderse como la administración de los recursos forestales tendiente a obtener el rendimiento óptimo de algún(os) bien(es) y servicios del bosque, minimizando el deterioro de éste(os) y de sus recursos asociados[4].

Con este marco quedó establecida la necesidad de desarrollar sistemas de manejo que se basen en los conceptos del MIF para la planeación y manejo de los recursos forestales. Atendiendo a esta necesidad se han desarrollado varias *metodologías de manejo forestal* que integran los conceptos del MIF. Entre ellas, se cuenta el *Sistema de Conservación y Desarrollo Silvícola* (SICODESI). El SICODESI se ha desarrollado dentro del acuerdo de cooperación científica y técnica entre México y Finlandia[5]. En su diseño se ha considerado el cumplimiento de los aspectos legales, así como la aplicación del MIF a bosques de coníferas de clima templado.

En la figura 4.1 se presenta un esquema que muestra los aspectos generales de la metodología del SICODESI, en él podemos apreciar que establece un conjunto de estudios que deben llevarse a cabo para realizar una planeación estratégica del manejo de los recursos forestales a largo plazo, estos son: estudios socioeconómicos, estudios dasométricos y estudios ambientales.

---

3 Publicada por el Diario Oficial de la Federación el 30 de marzo de 1986.

4 SARH, Subsecretaría Forestal. *Bases y principios del manejo integral forestal.* SARH México D.F. 1989.

5 SARH, & Universidad de Helsinki, *Documento principal del sistema de conservación y desarrollo silvcola*, México D.F. 1992.

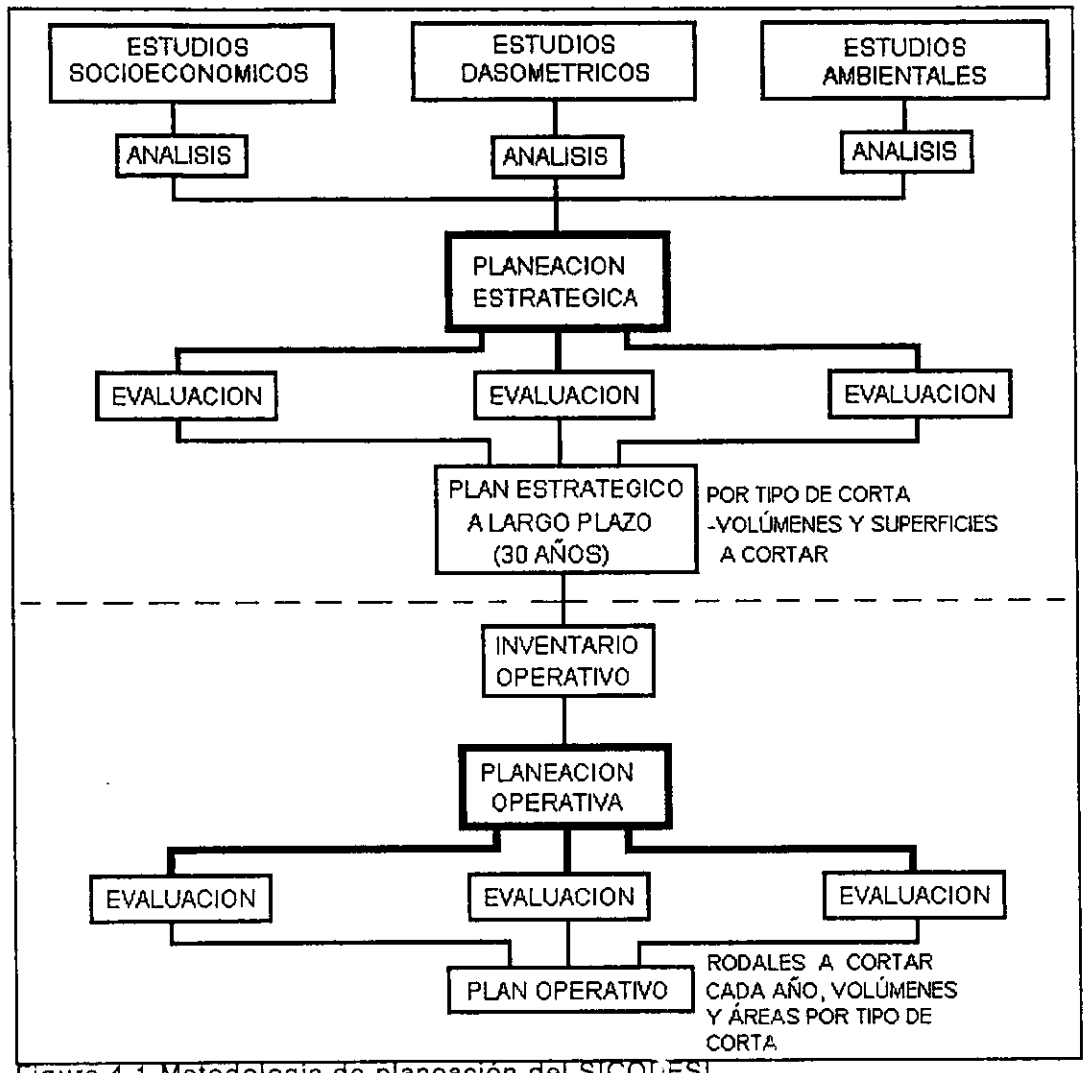


Figura 4.1 Metodología de planeación del SICODESI

Este esquema nos sirve para entender el entorno en el que va a funcionar el sistema que se pretende desarrollar, en este caso el entorno está constituido por un sistema que incorpora una metodología (SICODESI), la cual sigue unos lineamientos generales que para nosotros como analistas de sistemas, constituyen los requerimientos que debe satisfacer el sistema que se pretende desarrollar, en este caso el SIPLAMIF.

4.3.2 Descripción general del sistema

Basándose en la metodología de planeación del SICODESI y en la serie de estudios que éste contempla, podemos dividir el sistema en dos niveles, como se puede apreciar en la misma figura 4.1: La **planeación estratégica** y la **planeación operativa**. En la primera se evalúan las interacciones entre el sector forestal, los propietarios y otras actividades económicas, considerando el ecosistema; se analiza la situación actual del recurso bosque y se estiman los cambios futuros, elaborando un plan de producción forestal a largo plazo de 30 años. Este plan es utilizado en la planeación operativa como un objetivo respecto al nivel de producción, para producir un programa anual de cortas, en donde se especifican los *rodales*[6] a cortar, así como los volúmenes de madera a extraer en cada rodal.

A continuación se describe con mayor detalle estos dos niveles en que se descompone el sistema.

4.3.2.1 Planeación estratégica

La planeación estratégica tiene como principal objetivo la planeación de los aprovechamientos forestales a largo plazo, tomando en cuenta los requerimientos del equilibrio ecológico y de las actividades económicas y sociales que existen en el *área de planeación*[7], con el fin de obtener los mayores beneficios posibles de los bosques. Para ello son necesarias las actividades presentadas a continuación.

Estudios socioeconómicos y tecnológicos. Tienen por objeto integrar al proceso de planeación forestal todos los aspectos importantes de desarrollo del predio.

Objetivos:

* Establecer el marco socioeconómico y tecnológico en el que se desarrollarán los aprovechamientos forestales.

* Ajustar las actividades forestales a los objetivos de desarrollo del predio.

* Adecuar el aprovechamiento de los recursos forestales y la intensidad de su explotación a la cantidad de mano de obra, recursos disponibles y al mercado de productos forestales.

---

[6] Rodal es una parte del monte que tiene uno o varios rasgos en común: calidad de estación, composición de especies, edad, espesura, etc.

[7] Los expertos en el manejo de los recursos forestales le conocen a ésta área como el *predio*, se hace esta aclaración porque estos términos forman parte del dominio de la aplicación.

* Producir datos de costos, precios y productividad de los trabajos de explotación de los recursos forestales.

Estudios de impacto ambiental. La necesidad de estos estudios como parte del MIF es muy importante, dado que los tratamientos forestales[8] afectan su ambiente en diferentes formas, dichos estudios deben conocerse si se pretende manejar un bosque bajo bases firmes.

Objetivos:

* Describir la situación ambiental actual del predio y determinar los factores ambientales que posibilitan el deterioro del ecosistema.

* Usar esta información como apoyo a la clasificación del uso actual y potencial del suelo y en la evaluación del impacto ambiental.

Para realizar lo anterior se requiere de:

* Describir la situación actual

* Evaluar los impactos de las prácticas forestales al ambiente y ubicar las áreas críticas.

* Elaborar métodos para reducir los impactos negativos que las prácticas forestales pueden ocasionar en estas áreas.

* Reducir la intensidad de corta de aquellas áreas más susceptibles al deterioro por medio de restricciones.

Una vez ubicadas las áreas críticas y determinados los métodos para reducir los impactos negativos, se pueden integrar estos resultados al cálculo de potencialidad de corta para obtener un nivel óptimo de aprovechamiento sin incurrir en el deterioro ambiental.

La información se obtiene a través de trabajos de cartografía, de la interpretación de fotografías aéreas, de la información de inventarios estratégicos y de recorridos de campo en ciertas áreas.

Estudios dasométricos. En estos estudios se pretende por un lado tener la información suficiente para caracterizar la situación actual de los recursos y generar una serie de modelos dasométricos[9] para llevar a cabo la proyección de la masa forestal a través del tiempo. Para esto se elaboran dos niveles de

---

[8] Un tratamiento forestal es la aplicación de una corta a la masa arbolada

[9] Modelos para predecir altura, calidad de sitio, volúmen, predecir número y volúmen de renuevos, etc.

inventarios forestales, el inventario estratégico para la planeación a largo plazo (30 años) y el inventario operativo para la planeación a 5 años.

Inventario Estratégico. El objetivo del inventario estratégico es obtener suficiente información de los bosques, para la planeación estratégica (a largo plazo) y para la construcción de los modelos. Tiene como unidad de muestreo al *sitio*. En cada *sitio* de muestreo, para cada árbol se obtiene: especie, estrato, diámetro y parte dañada si fuera el caso. Se define una muestra de árboles y se eligen en forma sistemática, midiéndoseles las siguientes variables: altura total, altura de copa viva[10], el grosor de la corteza y la edad.

Cálculo de la potencialidad de los aprovechamientos. Consiste en establecer el nivel adecuado del aprovechamiento de los recursos forestales del predio, considerando las restricciones socioeconómicas y ambientales, así como las tecnológicas para obtener una posibilidad[11] persistente a largo plazo.

Como unidad de cálculo se utiliza al *sitio* del inventario estratégico o al *rodal*. Se requiere de la información obtenida en los estudios dasométricos y de ciertos supuestos que pueden ser cambiados de acuerdo a las condiciones del área de estudio.

4.3.2.2 Planeación operativa

Durante la planeación operativa se realizan una serie de actividades encaminadas a obtener un plan anual de producción, dichas actividades se describen a continuación.

Selección de la sub-área operativa. De la superficie total de planificación se elige una sub-área operativa en la cual se realizará el aprovechamiento.

Inventario operativo. Esta es una actividad en la que el planificador sugiere el tratamiento necesario y su prioridad de aplicación, se obtienen datos tanto a nivel predial, como a nivel rodal; como son superficie y volúmen a cortar de los diferentes tipos de cortas. Esta información se utiliza luego para hacer el cálculo operativo.

Cálculo operativo. Se realiza el cálculo de la situación actual, del desarrollo de los rodales, es decir, se calcula el incremento del rodal y en caso de haberse

---

[10] Distancia vertical entre el nivel del suelo y la porción donde se inicia la copa del árbol.

[11] La posibilidad es la cantidad de volúmen que se puede derribar periódicamente en un bosque, manteniendo su rendimiento lo más constante posible y de forma sostenida.

recomendado cortas, se describe su efecto sobre la distribución diamétrica, con un modelo de cortas, Esto se realiza para un período de cinco años.

Selección final de los tratamientos de los rodales. El objetivo es que en base a la información presentada se obtenga un programa que indique los tratamientos durante los próximos cinco años, manteniendo los volúmenes cortados más o menos a un nivel similar.

Elaboración del plan de manejo. Con base en la información obtenida anteriormente se elabora un plan de manejo que indica rodales a cortar cada año, rodales que no se intervendrán y los tratamientos complementarios que se necesitan para el desarrollo del bosque.

4.3.3 Análisis

En la descripción general del sistema se proporcionaron los requerimientos del SIPLAMIF, los cuales muestran la complejidad inherente al sistema. Es necesario capturar primero los conceptos más importantes que nos permitan limitar nuestro problema. Para esto identificamos los puntos funcionales del sistema, es decir, las transformaciones que el sistema realiza con su entorno. De esta manera tendremos los escenarios principales, que ilustran los comportamientos clave. Así revisando la descripción del sistema tenemos que los puntos funcionales son:

- Estudios socioeconómicos
- Estudios de impacto ambiental
- Estudios dasométricos (se realizan para un predio o para un conjunto de ellos)
- Realización de inventarios estratégicos y operativos
- Realización de una planeación del uso de los recursos del bosque a largo plazo (cálculo de potencialidad) y a corto plazo (cálculo operativo)

Al identificar estos puntos funcionales, estamos obteniendo una vista general del sistema. Cabe mencionar que los estudios anteriores se realizan a nivel predial y permiten un manejo integral de los recursos forestales (MIF).

Podemos representar esta visión general del sistema como en la figura 4.2 Encontramos entidades tangibles y eventos candidatos a formar clases de objetos. Como entes tangibles tenemos: predio, inventario estratégico, inventario operativo, modelos, y como eventos: estudios de costos, estudios de impacto ambiental, potencialidad y cálculo operativo.

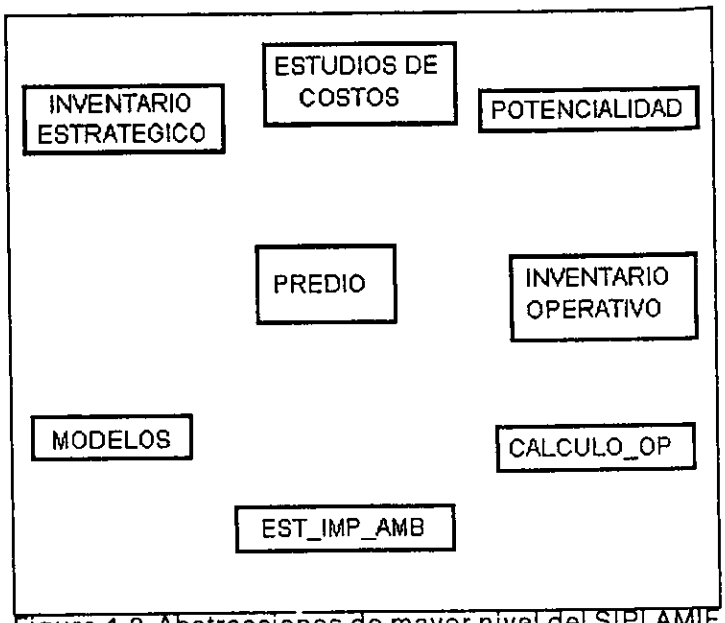


Figura 4.2 Abstracciones de mayor nivel del SIPLAMIF

Notese que hasta aquí solo hemos identificado las clases de objetos que son relevantes en el dominio del problema.

Esta fase de análisis no termina con la elaboración de una lista de clases candidatas, sino que hay que definir todavía sus atributos y responsalidades (segunda actividad del microproceso) y buscar las relaciones entre ellas (tercera actividad del microproceso).

4.3.4 Diseño

Es necesario establecer una estructura de clases que nos permita establecer las relaciones existentes entre ellas. En el nivel más alto de la estructura de clases, encontramos relaciones de uso más que relaciones de herencia.

Un predio es la unidad sobre la cual se realiza un estudio de Manejo Integral Forestal. Para llevar a cabo este estudio se requiere de una serie de elementos como son: realizar un inventario operativo, un inventario estratégico, estudios de costos, estudios de impacto ambiental, potencialidad y cálculo operativo. Estas además son las clases que la clase predio utiliza para su implementación.

En la figura 4.3 se muestra el diagrama de clases que muestra las relaciones entre las clases mencionadas arriba. También puede observarse la cardinalidad de esas relaciones de uso. Este diagrama nos representa una visión lógica de la estructura de clases del sistema.

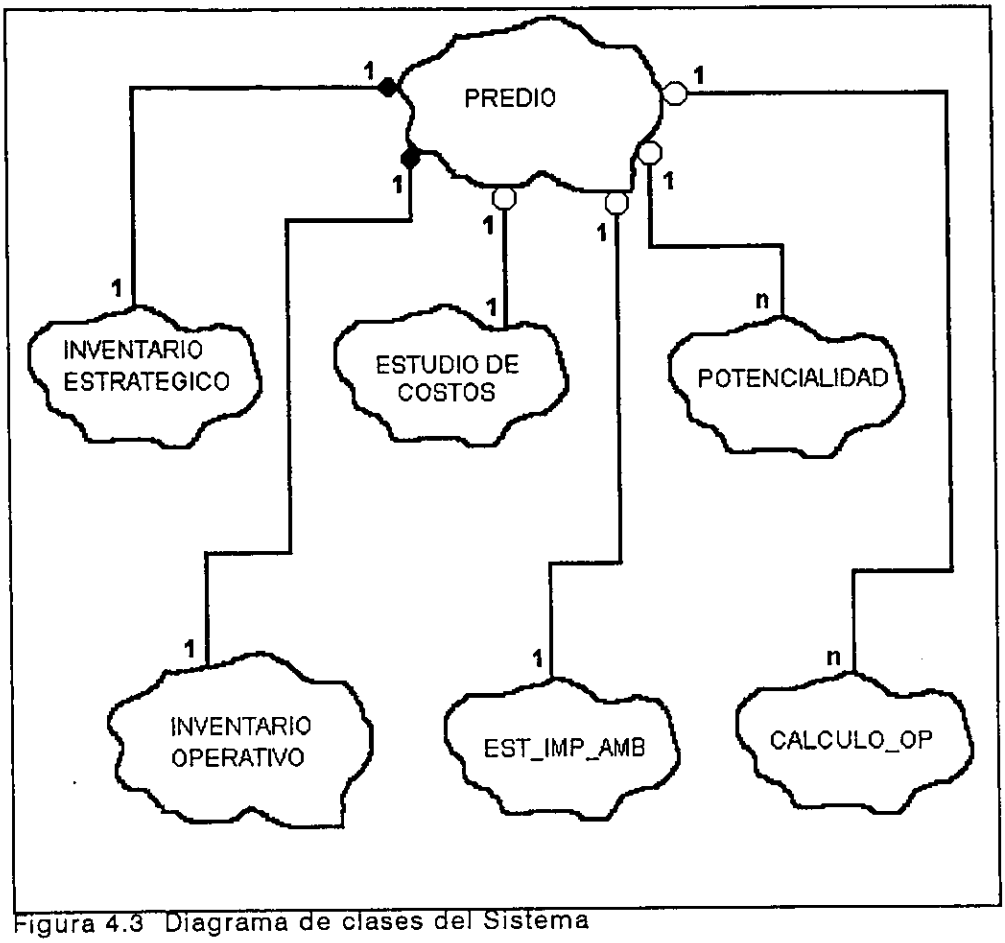


Figura 4.3 Diagrama de clases del Sistema

A continuación se analizan por separado las clases mostradas en el diagrama de clases anterior.

Inventario estratégico

En la figura 4.4 se muestra el diagrama de las clases inventario estratégico e inventario operativo, sus relaciones con otras clases así como su cardinalidad.

Un inventario estratégico, está compuesto por sitios temporales (sitio_temp) y sitios permanentes (sitio_per). Ambas clases son un tipo especial de las clase *sitio*, y por lo tanto heredan de ella tanto sus atributos como su comportamiento; la clase sitio es un *clase abstracta*[12]. Tanto la clase **sitio_temp** como la clase

[12] Una clase abstracta es la que no tiene instancias. Se escribe con la intención de que sus subclases añadan elementos nuevos a su estructura y comportamiento, normalmente implantando sus operaciones abstractas.

sitio_per, son visibles desde la interfaz de la clase inventario estratégico, y forman clases separadas porque difieren en algunos atributos.

La clase **sitio_temp** incorpora dentro de su interfaz a la clase **arbol_temp**, que es heredera de la clase árbol. Además la clase arbol_temp requiere para su implementación a la clase **grupos**, dado que entre los atributos de arbol_temp se encuentra especie, y los métodos que utilizan árboles necesitan tenerlos agrupados en conjuntos de especies. Por lo tanto se requiere tener una clase que nos permita conocer la agrupación de especies existente en el momento. Por otro lado la clase arbol_temp requiere para su implementación de la clase **volumen**.

Algo parecido ocurre con la clase **sitio_per**. La diferencia radica en que esta clase utiliza **arbol_per**. La clase arbol_per es diferente de la clase arbol_temp en una serie de atributos. Por ejemplo un árbol medido en un sitio permanente requiere de una ubicación espacial que nos permita volver a medirlo, mientras que uno temporal no.

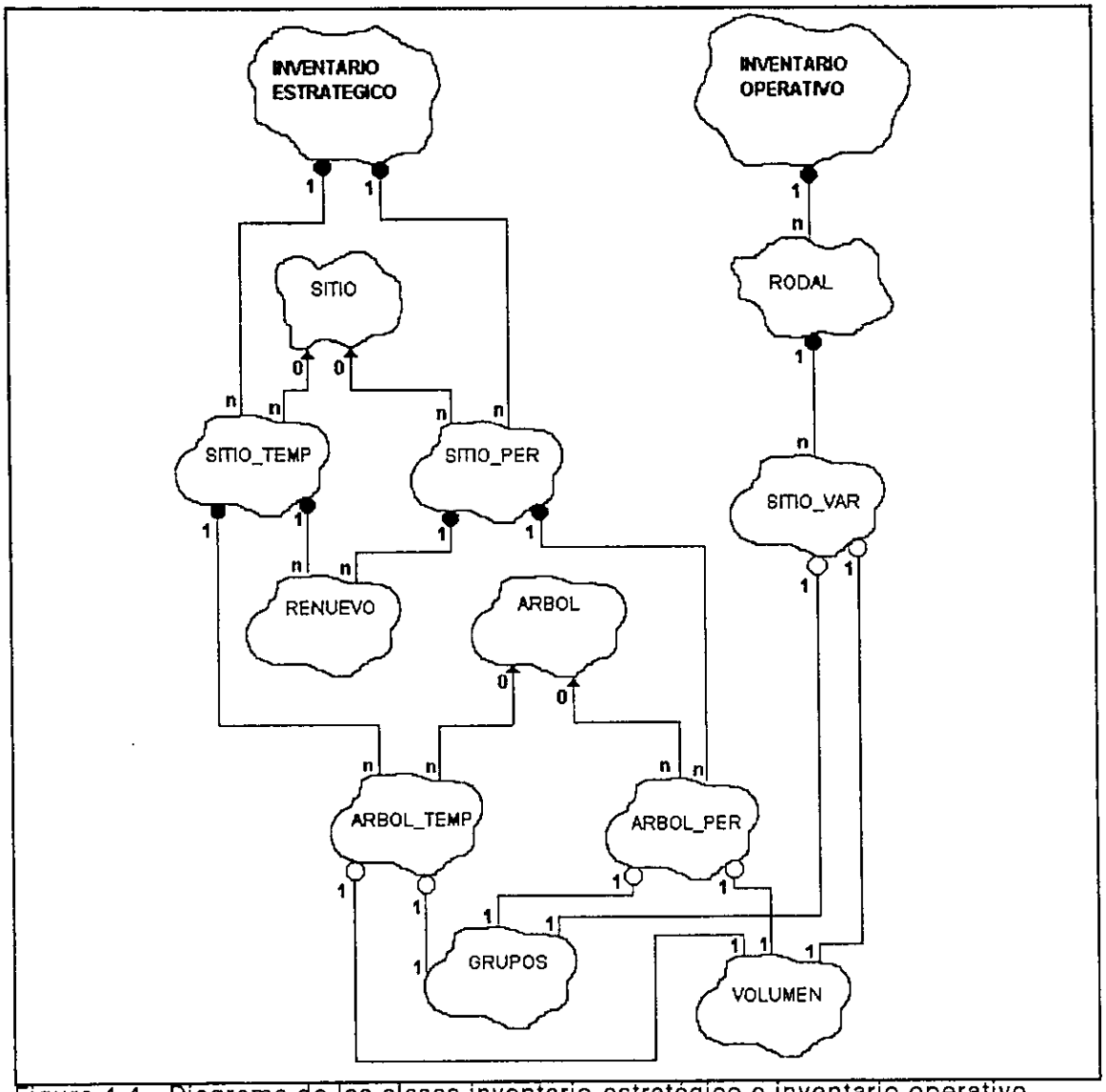


Figura 4.4 Diagrama de las clases inventario estratégico e inventario operativo

Por su parte la clase **inventario operativo** usa a la clase **rodal** dentro de su interfaz. La clase rodal a su vez usa la clase **sitio_var** (sitio de dimensiones variables), y esta última requiere para su implementación a la clase **grupos**.

A continuación se mencionan algunas características sobresalientes de estas clases.

sitio, sitio_temp y sitio_per. La clase sitio hereda sus atributos y su conducta, muchos de sus métodos pueden ser redefinidos, o incluso únicamente son declarados e implementados en sus subclases.

arbol, arbol_temp y arbol_per. La clase arbol es una clase abstracta, ya que no puede ser instanciada. Por lo tanto únicamente se utiliza para heredar sus atributos y su conducta a la clase arbol_temp y a la clase arbol_per.

renuevo. En la clase renuevo se captura información de los renuevos encontrados en los sitios de muestreo, tanto temporales como permanentes. De manera similar a la clase árbol, la clase **renuevo** utiliza la clase **volumen_ren**, que contiene los parámetros de la función de volumen, para el cálculo del volumen de renuevos.

De la clase **grupos** se obtiene el grupo al que pertenece una especie determinada. Además el administrador de los recursos forestales, durante la inicialización de los atributos de esta clase obtiene una lista de especies las cuales son agrupadas por él, de acuerdo a los hábitos de crecimiento y a la silvicultura de las especies. Las instancias de esta clase pueden ser formadas de otra manera, utilizando componentes basados en conocimientos. Para esto se requiere de una base de conocimientos de la silvicultura de las especies forestales encontradas en el país, y de una serie de reglas que nos permitan agrupar dichas especies. Estas reglas pueden ser manejadas como clases. De esta manera no es necesario la intervención de un administrador forestal en la agrupación de las especies.

Inventario operativo, rodal y sitio_var. La clase inventario_operativo es similar a la clase inventario estratégico. la diferencia radica en que el inventario operativo se realiza a nivel de rodales. En cada rodal se miden de acuerdo a una intensidad de muestreo ciertos sitios de dimensiones variables. En estos últimos se captura información del árbol medio, además se obtiene el área basal del sitio.

Modelos dasométricos

En la figura 4.5 se muestra el diagrama de la clase **modelos**. Esta clase usa para su implementación a la clase inventario estratégico, además existen modelos diferentes que heredan las características generales de ella.

Esta serie de estudios dasométricos se utilizan para realizar la proyección de la masa forestal através del tiempo, la información obtenida se utiliza para hacer los cálculos de potencialidad de los aprovechamientos y el cálculo operativo.

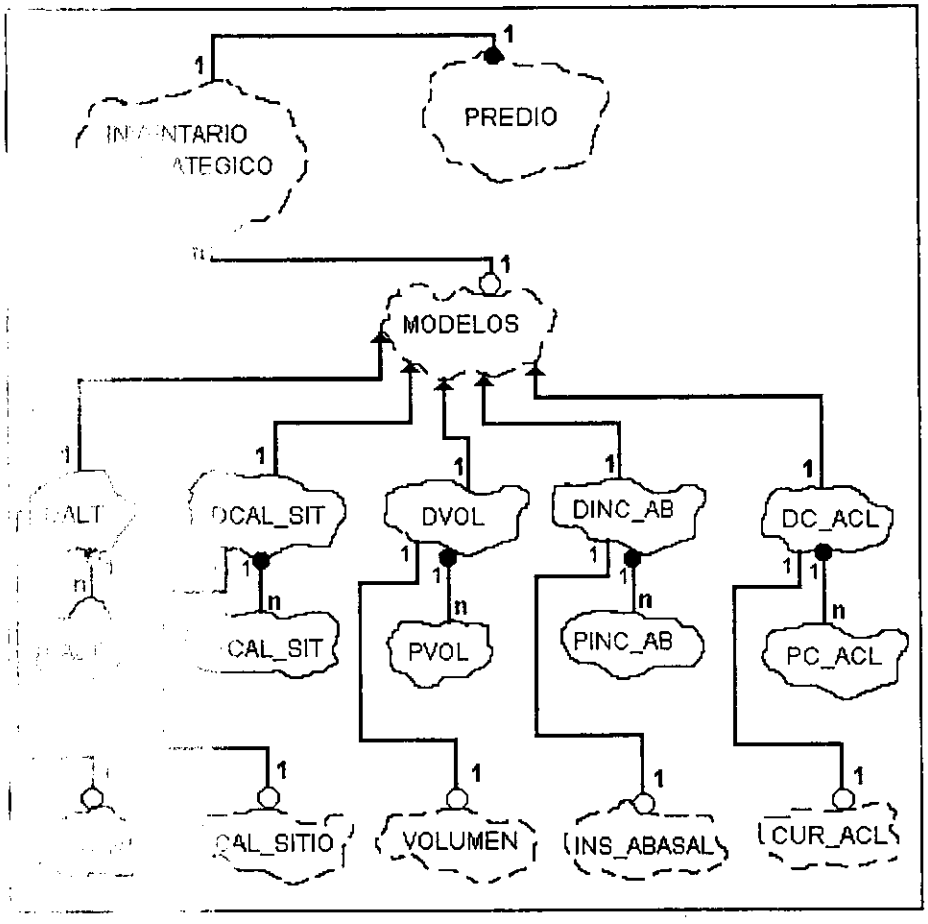


Figura 4.5 Diagrama de la clase modelos

...ergencia con respecto a los modelos es que alguna ecuación, por ...altura puede ser aplicable a cierta condición del bosque, ...ón del bosque requiere de una ecuación diferente. Dado ...os tiene la propiedad de polimorfismo es posible ajustar ...ra cada modelo, y emplear la ecuación correcta en el ... Para este ejemplo únicamente se considera que existe ...modelo.

Otra cuestión interesante es que todos los modelos dasométricos son ajustados con regresión, en algunos casos lineal y en algunos otros no lineal. Nosotros podríamos diseñar las clases necesarias para llevar a cabo una regresión, o bien utilizar un paquete estadístico que realize las operaciones que se requieran[13]. Para ello se debe diseñar una interfase que nos permita mandar al paquete estadístico tanto los datos como las instrucciones necesarias para que se realice el cálculo, además de obtener los resultados de la forma que se desee.

En seguida se presenta información relevante de algunas de las clases presentadas en la figura 4.5.

modelos. La clase modelos obtiene la información de árboles y renuevos necesaria para realizar el ajuste de cada uno de los modelos dasométricos. La información para tal ajuste es tomada del inventario estratégico, y puede ser de uno o varios predios.

dalt, palt y altura. Conforman el modelo de altura, en el cual la clase Altura usa para su implementación a la clase Dalt, en la cual se guardan los datos necesarios para el ajuste de la ecuación de altura. Además la clase Dalt tiene dentro de su interfase a la clase Palt, en la cual se guardan los parámetros del modelo de altura.

dcal_sit, pcal_sit y cal_sitio. Conforman el modelo de calidad de sitio, en el cual la clase Cal_sitio usa para su implementación a la clase Dcal_sit, que contiene los datos para el ajuste de la ecuación de calidad de sitio. Al igual que en el modelo anterior, la clase Dcal_sit tiene dentro de su interfase a la clase Pcal_sit, la cual tiene los parámetros del modelo de calidad de sitio.

Cabe mencionar que se deben considerar más modelos que los que se presentan en la figura 4.5, sin embargo su definición es similar, y para este ejemplo de diseño solo se consideran algunos de ellos.

Estudios de costos

La clase estudios de costos es otra de las clases encontradas en el nivel más alto del sistema. Esta clase es responsable de obtener los costos por metro cúbico del volumen de madera a cortar. Para ello se consideran los diferentes recursos con los que realiza la actividad de extracción. Además de la productividad con la cual se realizan los trabajos de extracción. Por otra parte es necesario contar con los

---

[13] Fernández, Villaseñor Yolanda. *Servicios de Manejo del conocimiento en ambientes de apoyo a la investigación científica*, IV Simposium Internacional sobre Inteligencia Artificial, Cancún, México Nov. 13-15 de 1991. pp. 343-349.

precios de los servicios técnicos, del derecho del monte y otros precios, agrupados en la clase **precios**.

Las clases relacionadas con la clase **estudios costos**, así como el tipo de relación y la cardinalidad se muestran en la figura 4.6. En seguida veremos algunas de las características de esta clases.

Estudios_costos. Debe obtener su información de la captura de datos por parte del usuario. Además tiene una serie de métodos que permiten capturar información de los recursos utilizados, así como los cálculos de los costos.

recurso. Esta clase captura información de todos los recursos utilizados en los trabajos de extracción para cada tipo de corta. Además nos permite obtener los costos generados por las actividades del aprovechamiento forestal de acuerdo al tipo de recurso. En los costos se incluyen gastos de materiales y salarios.

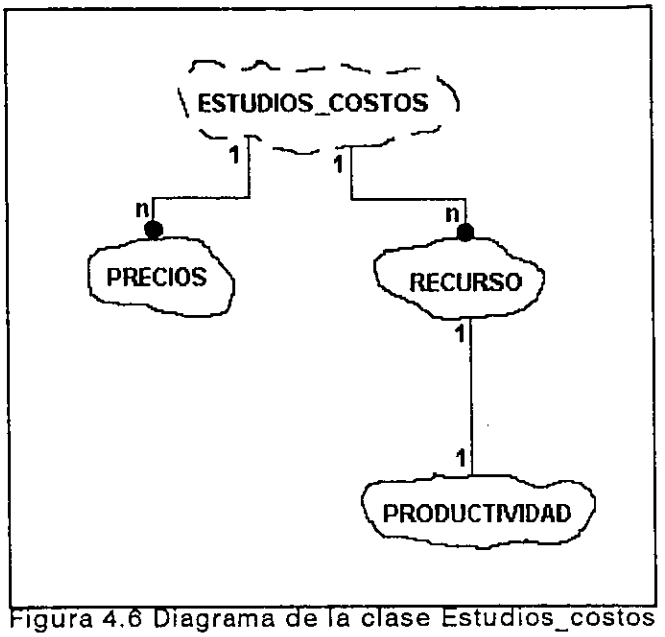


Figura 4.6 Diagrama de la clase Estudios_costos

Por otra parte utilizando las productividades registradas en las actividades de extracción realizadas en predios similares. Podemos obtener de acuerdo al volumen a extraer la cantidad de recurso utilizado.

productividad y precios. Estas clases exclusivamente obtienen su información del usuario en forma de captura, sin embargo, el usuario debe obtener la información requerida realizando algunos cálculos por fuera y de acuerdo al conocimiento que el tenga.

Estudios de impacto ambiental

La necesidad de estos estudios como parte de los MIF es muy importante, dado que los tratamientos forestales[14] afectan su ambiente en diferentes formas, las cuales deben conocerse si se pretende manejar un bosque bajo bases firmes.

[14] Un tratamiento forestal es la aplicación de una corta a la masa arbolada.

Los estudios de impacto ambiental están representados en el diagrama por la clase **est_imp_amb**. En ella se calculan las restricciones a tratamientos (restricciones a cortas) para evitar erosión; este cálculo puede ser para sitios o para rodales. Para realizar la tarea anterior la clase **est_imp_amb** usa un conjunto de clases, las cuales podemos ver en la figura 4.7.

Por otro lado es necesario realizar estudios cartográficos que permitan delimitar las áreas que deban ser cortadas y las que no. Para esto la clase Est_imp_amb se relaciona con las clases **mapa** y **analisis_mapas**.

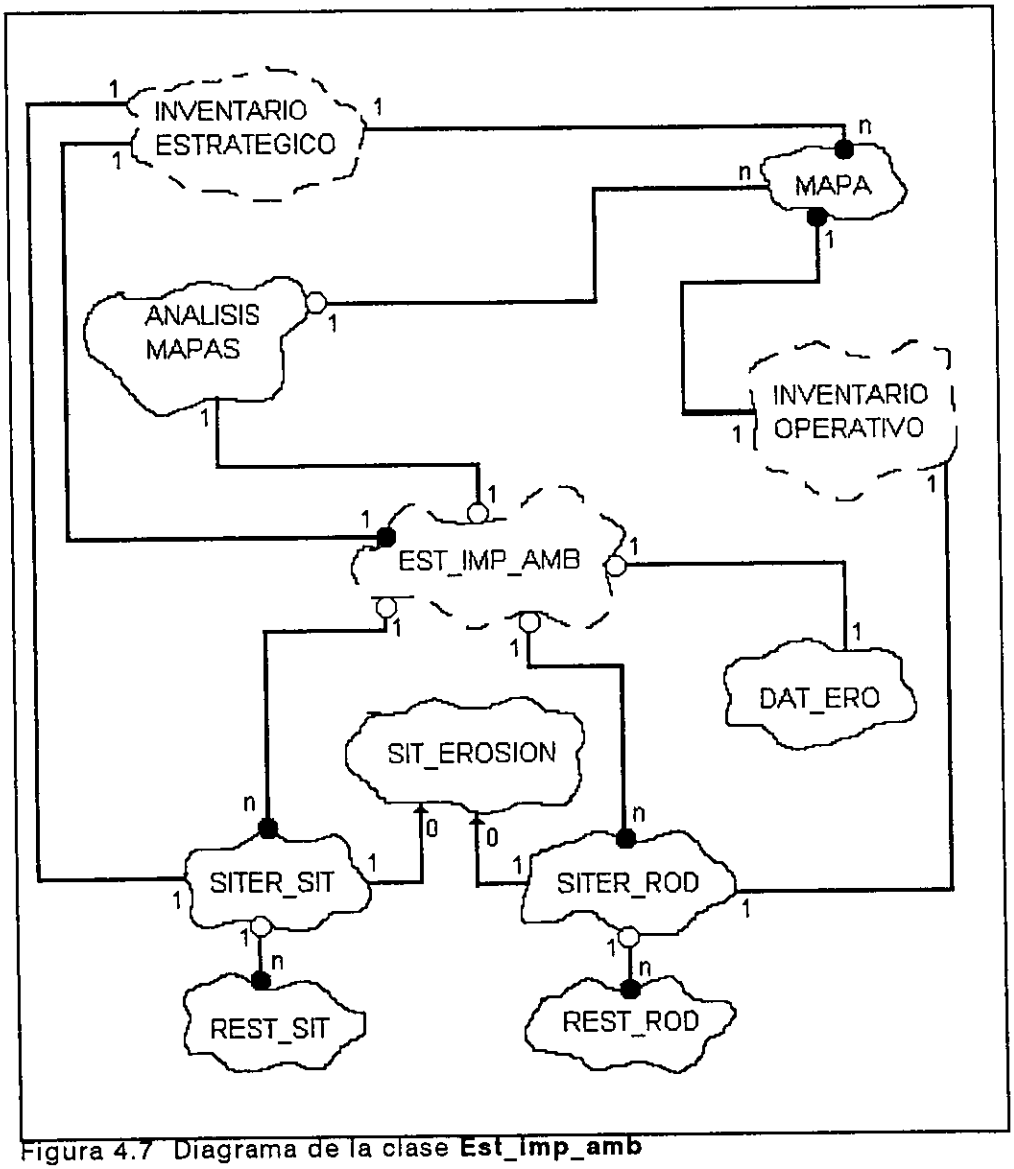


Figura 4.7 Diagrama de la clase **Est_imp_amb**

A continuación analizaremos brevemente las clases mostradas en la figura 4.7.

est_imp_amb. Esta clase se encarga de accesar los métodos de las clases con las que se relaciona para llevar a cabo: la captura de datos; el acceso a los datos de restricciones a cortas, tanto para el cálculo operativo como para el cálculo de la potencialidad de los aprovechamientos. Por otro lado también es responsable de accesar las clases que se relacionan con el análisis cartográfico.

dat_ero. En esta clase se captura información general necesaria para la determinación de las restricciones por erosión para algún *sitio* o *rodal*.

sit_erosion, siter_sit, siter_rod, rest_sit, rest_rod. La clase sit_erosion es una clase abstracta que contienen los atributos que son heredados a las clases que representan los *sitios de erosión* y los *rodales de erosión* (Siter_sit y Siter_rod respectivamente). Además en la clase Sit_erosion se define parte de la conducta de ambas clases. Por lo tanto la clase Sit_erosion no puede ser instanciada. La clase Siter_sit realiza los cálculos del índice de erosión para los *sitios*, para realizar esto utiliza información del inventario estratégico. Por otro lado esta clase usa a la clase Rest_sit, para almacenar las restricciones de cada sitio, que restringen la corta. Esta información será utilizada en el cálculo de la potencialidad de los aprovechamientos. De manera similar actúa la clase Siter_rod, pero usa información del inventario operativo y guarda las restricciones en la clase Rest_rod. Estas últimas son utilizadas en el cálculo operativo.

mapa. El análisis de mapas como información ha llegado a ser una parte integral en la planeación del manejo de los recursos naturales. De hecho la tecnología de sistemas de información geográfica nació por parte de las aplicaciones especializadas.

Es factible representar a un mapa como una clase con sus propiedades y conductas y también para realizar una interfase con un sistema externo (un GIS colaborador) haría falta manejar otras clases que no se describen aquí. De hecho es tan amplio el tema que podría considerarse como una investigación particular.

La clase mapa tiene atributos generales como titulo, escala, fecha, coordenadas límites, entre otros. También tiene parámetros descriptivos, como por ejemplo, suelo, vegetación, etc. Y un conjunto de categorías para cada parámetro descriptivo. por ejemplo para vegetación, tendríamos varias categorías: bosque tropical, bosque de coníferas, etc.

Como métodos de esta clase están la creación, la sobreposición y la impresión de mapas, el cálculo de la distancia entre dos puntos y algunas operaciones particulares a la aplicación como la rodalización. La rodalización consiste en encontrar los límites de superficies con características topográficas, de vegetación, etc., comunes.

Cálculo de la potencialidad

El cálculo de la potencialidad de los aprovechamientos, es realizado por la clase potencialidad. En la figura 4.8 se muestra el diagrama de ésta, las clases con las que se relaciona y la cardinalidad de dichas relaciones.

En la clase potencialidad se obtiene la situación actual de los recursos en cuanto a volúmenes y superficies. Además se proyecta el crecimiento de la masa forestal a través del tiempo en períodos de cinco años. En cada período se estiman los volúmenes y superficies a cortar por tipo de corta. Excluyendo las áreas no aptas para la corta, o bien restringiendo la intensidad del tratamiento.

Por otra parte se determinan los costos que corresponden con el volumen de madera propuesto para extracción. Estas tareas se llevan a cabo gracias a la colaboración de las clases que se muestran en la figura 4.8. A continuación se analizan algunas de las características de dichas clases.

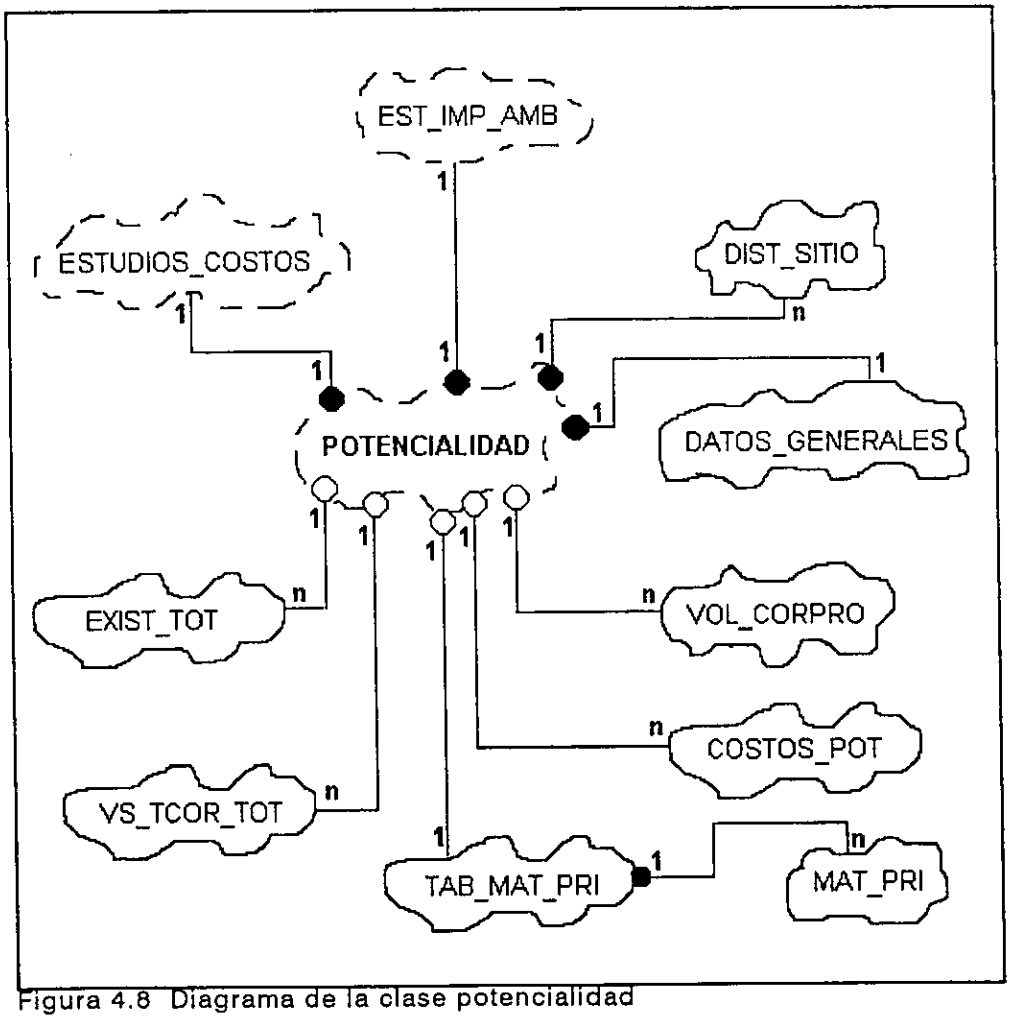


Figura 4.8 Diagrama de la clase potencialidad

potencialidad. La clase potencialidad es la encargada de coordinar las actividades para obtener la planeación estratégica de los recursos forestales. Su principal fuente de información son los sitios de distribución representados por la clase **dist_sitio**, los estudios de costos y los estudios de impacto ambiental. Las cortas en cada *sitio* se realizan de acuerdo a las condiciones del bosque reflejadas en el inventario estratégico, ordenandose los *sitios* en base a algún criterio para cortar.

datos_generales. Esta clase contiene información de los supuestos generales para realizar el cálculo de la potencialidad. Los supuestos son capturados por el usuario.

exist_tot, vs_tcor_tot, vol_corpo. En estas clases se almacenan los resultados tanto del volumen de las existencias como de la superficie, así como los volúmenes de corta por tipo de tratamiento y por tipo de producto.

costos_pot. En esta clase se almacenan los resultados de costos y beneficios obtenidos del volumen de cortas propuesto para su extracción.

tab_mat_pri y mat_pri. La clase Tab_mat_pri contiene una tabla de materia prima, en la cual por cada grupo de especies, se tiene tanto el tipo de producto (celulosa, aserrio largas dimensiones y cortas dimensiones) como el porcentaje del volumen total del árbol con esas características (diámetro, altura, especie). La tabla de materia prima usa una lista de clases mat_pri.

En la figura 4.9 se muestra el diagrama de la clase **dist_general**. Ella hereda sus atributos y su conducta a las clases **dist_sitio** y **dist_rodal**. Estas dos últimas son usadas por la clase **potencialidad** y la clase **calculo_op** respectivamente. La clase **dist_general** se analiza por separado debido al tamaño e importancia de sus clases herederas.

Los datos necesarios para realizar las distribuciones de los *sitios* y de los *rodales* se obtienen del inventario estratégico y del inventario operativo respectivamente. También se requiere de los modelos dasométricos para realizar las proyecciones, como algunos resultados son por tipo de producto se utiliza la clase **tab_mat_pri**.

Además la clase **dist_general** utiliza tres clases para guardar sus resultados. en la figura 4.9 se pueden observar las relaciones descritas entre las clases, así como la cardinalidad de las relaciones.

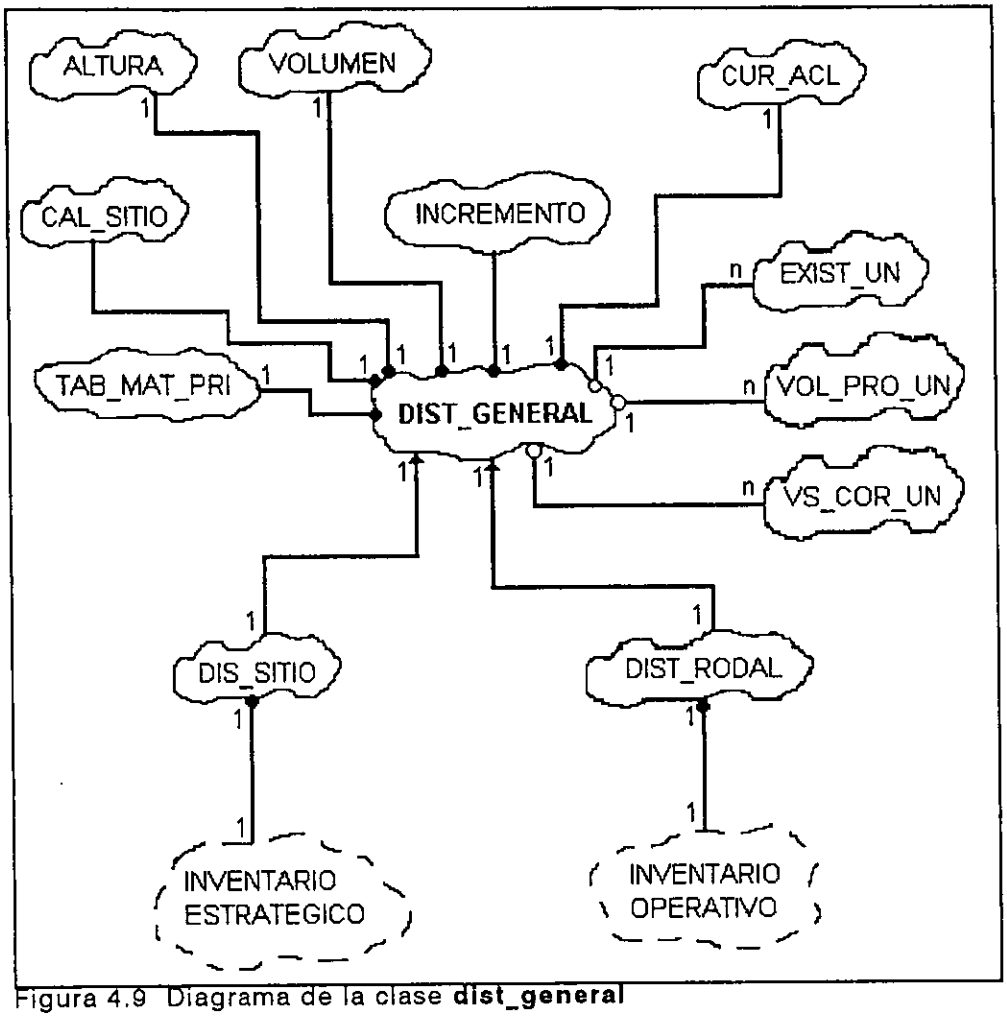


Figura 4.9 Diagrama de la clase **dist_general**

A continuación se analizan brevemente las características de estas clases.

dist_general. Esta es una clase abstracta que hereda sus atributos a las clases dist_sitio y dist_rodal.

exist_un, vol_pro_un y vs_cor_un. En estas clases se guarda para cada distribución: el volumen, y la superficie por clase de desarrollo, clase de edad y grupo de especie. Además se almacena el volumen de corta y la mortandad por tipo de producto y grupo de especie. También guardan el volumen y la superficie a cortar por tipo de tratamiento. Estas clases tienen procedimientos para inicializar sus parámetros, y para accesarlos.

dist_sitio. En esta clase se almacena información de la distribución de los sitios. La información es tomada del inventario estratégico. Cabe señalar que esta clase representa una parte central para el desarrollo del cálculo de la potencialidad de los aprovechamientos, pues como pudo observarse los cálculos se realizan con los sitios de distribución.

Los modelos dasométricos son usados por esta clase para conocer el desarrollo y estructura de la masa forestal a través del tiempo. Además se tienen procedimientos para simular la corta en las distribuciones que se realizan de acuerdo a ciertos criterios, según el tipo de tratamiento.

dist_rodal. En esta clase se almacena información de la distribución de los rodales. Su estructura es similar a la de distribución de los sitios, la diferencia radica en que se utilizan rodales para el cálculo de la distribución y no sitios estratégicos. Esta clase es usada en el cálculo operativo.

Cálculo operativo

El cálculo operativo es realizado por la clase **calculo_op**. En la figura 4.10 se muestra el diagrama de ésta, las clases con las que se relaciona, además de la cardinalidad de dichas relaciones. En la clase **calculo_op** se obtiene la misma información que en la clase potencialidad. La diferencia entre ambas estriba en que en el cálculo operativo los períodos son anuales. Además las cortas en las distribuciones se realizan de acuerdo al tratamiento prescrito en campo, y en el tiempo prescrito por el manejador de los recursos.

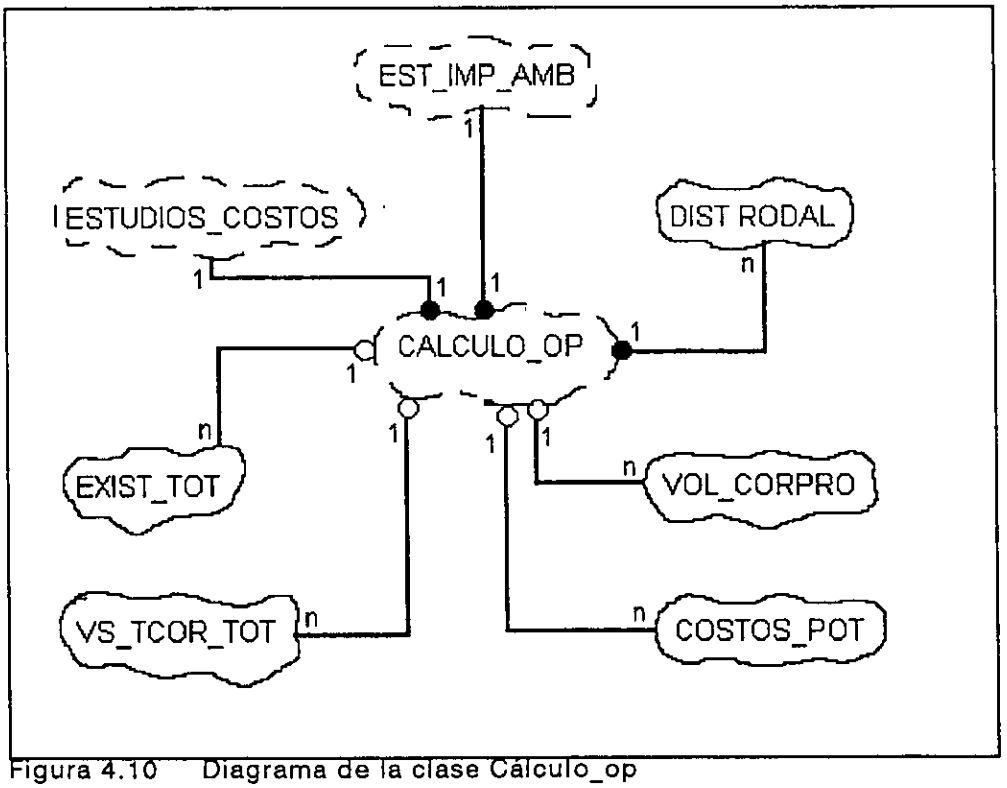


Figura 4.10 Diagrama de la clase Cálculo_op

La prescripción de la corta se realiza directamente en el inventario operativo en campo. Sin embargo el año en el que se va a realizar la corta es puesto por el

usuario. Esta última actividad se realiza tomando en cuenta, que los rodales a intervenir (a cortar) se encuentren juntos para facilitar las tareas de extracción. Además de que los rodales deben ser accesibles, y tener una cierta distancia a un camino, etc.

Sin embargo, esta tarea se puede realizar por el sistema, de acuerdo a ciertas reglas que coloque el usuario y haciendo un análisis de la información por medio de mapas, utilizando componentes basados en conocimientos. Cabe aclarar que esta es una tarea bastante compleja.

La clase **cálculo_op** es la encargada de coordinar las actividades para obtener la planeación operativa de los recursos forestales. Su principal fuente de información son los rodales de distribución representados por la clase **dist_rodal**, los estudios de costos y los estudios de impacto ambiental. Se debe aclarar que las cortas que se realizan en los rodales son de acuerdo a la prescripción en campo del tipo de tratamiento, y el año de aplicación.

4.3.5 Guía para la implementación

Debido a que el enfoque de este trabajo ha sido orientado hacia el análisis y diseño del software orientado a objetos, más que a la programación, no se presenta la implementación del sistema.

Actualmente se esta construyendo un prototipo demostrativo con la finalidad de que los usuarios del sistema opinen acerca de la interfase que tendrá el sistema final.

Cabe hacer notar que como parte de las actividades del Acuerdo de Cooperación Científica y Técnica en Materia Forestal entre México y Finlandia se pretende realizar la implementación completa de este sistema, considerando al paradigma orientado a objetos.

En la siguiente sección se discute inicialmente sobre el lenguaje utilizado para la creación del prototipo y las herramientas utilizadas.

4.3.5.1 Herramientas de software y equipo utilizado

Durante la fase de implementación es importante tener lenguajes orientados a objetos y herramientas de desarrollo que soporten el paradigma de diseño orientado a objetos, con la finalidad de tener una transferencia fluida entre las clases que existen en el diseño y las clases que se pretende constituyan el sistema.

Si bien el paradigma de diseño orientado a objetos, es independiente del ambiente especifico de implementación que se utilice, existen herramientas de implementación que afectan al diseño tanto como al desarrollo del proceso. Esto es, cuando se utiliza un lenguaje de programación que no es orientado a objetos, deben hacerse adecuaciones, es decir se debe eliminar la herencia, así como otras de las características de la orientación a objetos, *ausentes en los lenguajes tradicionales.*

Por lo anterior se consideró importante utilizar en la implementación un lenguaje que soporte el paradigma orientado a objetos. Aquí se utilizó el C++, con el compilador de Borland C++ 3.0. Junto con este compilador se utilizaron herramientas para la programación de la interfase con el usuario (TURBO VISION) y otra para el manejo de bases de datos (CODEBASE 4.5).

Turbo Vision es una biblioteca de clases de objetos de Borland, que permiten crear la interfase con el usuario dentro del ambiente DOS. Estas bibliotecas contienen una serie de clases para el manejo de ventanas, Menús, Cajas de diálogos, "Scroll_Bars", y soporte para ratón. Con la ventaja de que las bibliotecas son basadas en código orientado a objetos y pueden fácilmente extenderse y derivarse nuevas clases.

CodeBase es una biblioteca de C para el manejo de Bases de datos. Su compatibilidad con Dbase IV, Clipper, y Fox Base permite crear y accesar directamente archivos *.DBF, así como archivos indexados. Adicionalmente para los programadores de C++, existe una extensión de esta librería en la cual existen clases de objetos que son fácilmente manipulables.

El hardware utilizado durante el desarrollo del prototipo es una PC IBM 386 con 2 Mb. de memoria RAM, y con disco duro.

Tanto el software como el hardware se seleccionaron considerando que los usuarios del SIPLAMIF, las UCODEFO's (Unidades de Conservación y Desarrollo Silvícola) cuentan con pocos recursos para la compra de equipo de cómputo. Las UCODEFO's son organizaciones que se encargan del manejo de los recursos naturales en los estados de la república.

Esta aplicación nos ha presentado un panorama sobre el diseño de sistemas complejos en el área de manejo forestal, y sobre el manejo de grandes volúmenes de información de origen variado: encuestas socioeconómicas y de precios, inventarios con información general de *sitios* y *rodales*, así como información obtenida de mapas y fotografías aéreas. Por otro lado también observamos que se deben manejar los conocimientos de expertos en manejo forestal, silvicultura, ecología, geografía, etc.

Una conclusión que nos puede dejar este ejemplo con respecto al análisis y diseño de este tipo de sistemas se basa en el hecho de que para obtener los

beneficios del paradigma de la orientación a objetos se debe poner especial atención en la definición precisa de cada una de las clases, así como de las relaciones que deben existir entre ellas. Si no se hace de esta manera, posiblemente al intentar alguna extensión o modificación al sistema sea necesario hacer una reorganización masiva de dichas clases.

A continuación se describirán las principales tendencias del software posibilitadas por las ventajas de la adopción de la programación orientada a objetos.

4.4 Tendencias del software orientado a objetos

Las tendencias posibilitadas por las ventajas que ofrece la programación orientada a objetos con respecto al manejo de la complejidad y al mejoramiento de la productividad durante el proceso de desarrollo de software, van dirigidas hacia la reutilización de código depurado y comprobado contenido en bibliotecas de clases a las que el usuario final podrá accesar para crear sus aplicaciones adaptándolas a sus propias necesidades.

Una biblioteca de clases es una colección de clases genéricas que pueden adaptarse para una aplicación determinada. Cuanto más coincide una biblioteca de clases con una aplicación, menos modificación necesita.[15]

Podemos hablar de dos tipos de bibliotecas de clases:

Por un lado, tenemos las que contienen unidades fundamentales para la construcción de aplicaciones triviales, como cadenas, pilas, listas enlazadas y objetos básicos como botones, rectángulos o ventanas. Pueden utilizarse prácticamente por cualquier aplicación de bajo nivel, por ejemplo, las bibliotecas para construir interfases gráficas de usuario ya están disponibles, tal como *Common View* de Intuitive solutions.

Object works es un entorno que puede estar basado en C++ o en Smalltalk. La versión de C++ proporciona una interfase con UNIX, comprobación estricta de tipos, depuración a nivel de fuente y admite herencia múltiple. La versión Smalltalk 80 contiene una biblioteca de más de 300 clases y varios miles de métodos. Se puede ejecutar ya sea en PC DOS o en estaciones de trabajo UNIX.[16]

En el otro lado tenemos las bibliotecas de clases que proporcionan un marco estructural completo para una categoría de aplicaciones que caen dentro de un

---

[15] Winblad, L. Ann. *et. al. Software Orientado a Objetos*, Addison-Wesley Iberoamericana S.A. Wilmington, Delaware, E.U.A., 1993. pp. 78-79.

[16] Graham, Ian *Métodos orientados a objetos*, segunda edición, Addison-Wesley Iberoamericana, Wilmington, Delaware, E.U.A. 1996. pp. 100.

dominio específico. Estas bibliotecas contienen clases con un mayor nivel de complejidad, es decir, que contienen objetos complejos con un mayor número de interrelaciones.

El principal objetivo de la utilización de bibliotecas de clases, es a largo plazo, es reducir el costo de desarrollo de sistemas mediante la reutilización de clases especializadas. La doctora Hanna Oktaba[17], respecto al impacto económico de la utilización de la orientación a objetos, apunta que éste no puede medirse en base a la construcción de un solo sistema y que la ganancia del uso de esta metodología se percibe a largo plazo, donde la reutilización de los componentes capitalizados se vuelve significativa, y cita como ejemplo un estudio elaborado por los laboratorios AT&T en E.U., que reportó que un 80% de código de las nuevas aplicaciones, en el área de diseño de redes de computadoras, fue reutilizado después de cuatro años de ser adoptada una metodología orientada a objetos.

4.5 Las herramientas CASE como soporte al desarrollo orientado a objetos

Una de las principales tendencias van encaminadas hacía la construcción de herramientas CASE que soporten el desarrollo orientado a objetos. En los últimos años han surgido nuevas herramientas CASE que soportan las notaciones y metodologías de análisis y diseño más conocidas como: OOA/OOD de Coad y Yourdon, Shlaer y Mellor; OMT de Rumbaugh, Martin y Odell, Booch, etc. La mayoría de estas herramientas han surgido para soportar estas nuevas metodologías; por ejemplo, ObjectMaker de Mark V Systems, Intelligent OOA de Kennedy Carter, OOATool, OODTool de Object International Inc.

La mayor parte de estas herramientas funcionan bajo windows en ordenadores personales o en máquinas con sistema operativo UNIX.

Otra aproximación hacia las CASE OO se presenta en que los fabricantes de SABDOO (Sistemas Administradores de Bases de Datos Orientadas a Objetos) también han empezado a distribuir entornos de programación y herramientas CASE para el diseño de sistemas basados en sus productos, por ejemplo Ontos, distribuidor de ONTOS DB, presenta una herramienta de desarrollo interactivo denominada Studio, y una herramienta de diseño de esquemas llamada DB Designer.

En cuanto a los fabricantes de entornos y lenguajes de programación también han empezado a abordar temas relativos al análisis y diseño; en este sentido se puede mencionar, por ejemplo, EiffelCase de ISE y Object Studio de ENFIN.

---

[17] Oktaba, Hanna. artículo: El impacto de la ingeniería de software en la programación orientada a objetos, México, UNAM, 1991. p. 9

Una atención especial merecen las herramientas CASE adaptables[18], algunas de las cuales permiten al usuario definir su propia metodología y los símbolos de la notación a utilizar, se les conoce a este tipo de herramientas como meta-CASE. Un ejemplo lo constituye Mark V Systems que distribuye junto con su herramienta ObjectMaker un *Tool Development Kit* que permite hacer estas tareas.

Según algunos expertos, los líderes de del mercado CASE del futuro pueden provenir de la comunidad de la inteligencia artificial.

4.5.1 Las herramientas del futuro

La herramientas CASE actuales[19] presentan una serie de limitaciones con respecto al soporte a la orientación a objetos. En general, puede señalarse que aquellas herramientas que soportan muchas notaciones no consiguen realmente ayudar en la aplicación de una metodología con todo su proceso, sino que generalmente se quedan en un nivel exclusivamente gráfico. Por el contrario las que se centran en una sola metodología, consiguen recoger prácticamente toda su semántica (significado) y ayudar al desarrollador en la validación de los sistemas, además de generar código de mayor calidad.

La mayor parte de las herramientas CASE generan código C++; algunas pueden llegar a generar hasta el 60% del código del sistema. Una limitación importante en este sentido es que muy pocas generan código en otro lenguaje que no sea C++, como por ejemplo, Smalltalk o Eiffel.

La próxima generación de herramientas o como podríamos llamarle las herramientas del futuro, además de resolver las limitaciones que han señalado deberán contemplar entre otras las siguientes características:

- Una mayor integración entre los entornos de ejecución y las herramientas CASE.
- Un entorno experto que sea capaz de guiar al usuario durante todo el proceso de desarrollo. Por ejemplo, que durante la fase de análisis sea capaz de interpretar los requisitos y, dada una descripción textual, crear o proponer un diagrama de objetos inicial.

---

[18] Piattini, Mario G. *et. al. Elementos y Herramientas en el Desarrollo de Sistemas de Información, una visión actual de la tecnología CASE.* Addison-Wesley Iberoamericana S.A., E.U.A. 1995. pp. 300.

[19] En el apéndice se dan a conocer algunas de las herramientas CASE más difundidas en el mercado español y se hace una descripción a modo de ficha, que incluye año de la primera versión, versión actual, principales módulos, plataformas, fases del ciclo de vida soportadas así como otras características más.

- Ofrecer una mayor visión dinámica de la colaboración entre las clases
- Evaluar el rendimiento, teniendo en cuenta datos de la arquitectura hardware destino.
- Integración con repositorios de metadatos (que se adapten a estandares) relativos a las fases de análisis y diseño.

Pero sobre todo si se quiere conseguir las ventajas en cuanto a productividad y calidad que pregona la orientación a objetos, existe un elemento que se debe contemplar y que actualmente esta muy poco desarrollado; la **reutilización** no solo de código sino de diseños completos.

El soporte a la reutilización en herramientas CASE OO, consistirá en que la propia herramienta ayudará a identificar objetos candidatos, lo que requiere un esquema de clasificación completo y consistente, recuperando los objetos seleccionados, en primer lugar de modo local, y si es necesario, expandiendo la búsqueda utilizando algún método de acceso remoto a objetos[20]. Luego la herramienta CASE presentará al usuario el objeto que encaja mejor con la solución, calculando su costo y pudiendo con la ayuda del desarrollador refinar el objeto para adaptarlo mejor a la solución.

4.6 Aplicaciones en Inteligencia Artificial (IA)

En la IA existe una gran variedad de aplicaciones en las cuales se pretende que las máquinas realicen cosas que podrían requerir de la inteligencia humana. Dentro de tales aplicaciones tenemos el desarrollo de sistemas expertos que abarcan diversas áreas tales como: análisis, planeación, estimación, diagnóstico y control, que se aplican dentro de la industria en ramas como la manufactura, transportación, entrenamiento y servicios financieros.

Los sistemas expertos (o sistemas basados en el conocimiento), pretenden utilizar algunas de las lecciones aprendidas de la IA para resolver problemas prácticos. En particular los sistemas expertos pretenden simular el comportamiento de un "experto" en algún tema muy concreto.

Por ejemplo, los sistemas expertos para la capacitación en la industria[21]; en ellos se mide el rendimiento de una persona en base a la respuesta a preguntas hechas y se mide el tiempo que tarda el individuo en responder correctamente. Si el tiempo de respuesta es corto, entonces quiere decir que el individuo va progresando en su entrenamiento y el sistema incrementa la velocidad y

---

20 *Ibidem*, p. 304.

21 Se puede encontrar información más detallada de éste caso de estudio en: Lindsay, Susan *Practical aplicatios of expert systems*, QED information sciences, U.S.A. 1988. pp. 83-98.

complejidad del entrenamiento; por el contrario; si el individuo tarda más tiempo en sus respuestas, entonces el sistema reduce la velocidad y complejidad del entrenamiento. Este tipo de sistemas permiten que la capacitación sea individualizada.

El desarrollo de este tipo de aplicaciones obviamente es complejo, por lo que las técnicas de orientación a objetos proporcionan el apoyo necesario para analizar y diseñar este tipo de aplicaciones.

El lenguaje que se ha utilizado para la investigación en IA ha sido por mucho tiempo **LISP**[22], posteriormente éste se ha extendido para incorporar los conceptos de clases, funciones genéricas, métodos y herencia múltiple; resultando el Common Lisp Object System o **CLOS**,como se le conoce comunmente.

En CLOS los objetos son definidos como clases, los métodos que ejecutan operaciones sobre clases específicas son seleccionados, organizados y ejecutados por funciones genéricas. Este enfoque de funciones genéricas distingue a CLOS de los lenguajes de programación orientados a objetos como Smalltalk y Flavors que invocan los métodos por medio del paso de mensajes. En estos lenguajes, una operación se activa enviando un mensaje a una instancia de la clase. Si la clase contiene un método para manipular el mensaje, entonces la operación deseada se ejecuta[23].

4.6.1 Tendencias en el desarrollo de sistemas expertos

Los sistemas expertos constan de una *base de conocimientos*, que contiene los objetos y reglas correspondientes al conocimiento especializado de la aplicación; además tienen un *motor de inferencia* que es un programa que aplica el conocimiento que esta en la base de conocimiento a los datos que se le presentan al sistema. Si a un sistema experto se le priva de su conocimiento específico de la aplicación y se le dan los medios a los desarrolladores para "insertar" alguna otra base de conocimiento, entonces se dice que se trata de un *shell* de un sistema experto. Un shell es un entorno de programación que eleva la productividad de los constructores de sistemas expertos.

La principal tendencia que se esta dando en esta área es precisamente el desarrollo de entornos de programación o shells que contengan métodos de

---

22 LISP es un lenguaje de programación de alto nivel utilizado en IA y que se basa en el procesamiento de listas.

23 Se puede consultar más sobre la programación en CLOS; la definición de clases, instancias, funciones genéricas y métodos en: Lawless, Jo A. *et. al. Undestanding CLOS: the Common Lisp Object System*, digital press, U.S.A. 1991. pp. 1-3

inferencia y métodos para el tratamiento de incertidumbre y también alguna forma de añadir conocimiento. Los siguientes son algunos ejemplos de shells de sistemas expertos con características orientadas a objetos:

ADS de Aion Corporation. Es un entorno de desarrollo de sistemas expertos con encadenamiento progresivo[24], basado en reglas, pero las últimas mejoras proporcionan apoyo para el encapsulamiento y clases que pueden heredar tanto atributos como métodos.

KBMS de Aion Corp. Es un entorno con fuerte acoplamiento con sistemas de bases de datos, como DB2, con características orientadas a objetos en cuanto a que ofrece herencia de métodos y apoyo al encapsulamiento y admite herencia múltiple.

[24] El encadenamiento progresivo (forward chaining) es un método de búsqueda o de inferencia, que comienza con uno o más hechos, y sigue hacia adelante una cadena de reglas para obtener una o más consecuencias de los hechos.

CONCLUSIONES

Las aplicaciones actuales manejan información de diferente origen y para desarrollarlas se necesita conocer las técnicas orientadas a objetos ya que éstas nos proporcionan los mecanismos capaces de representar las complejas relaciones que existen entre los diferentes componentes de un sistema.

Con respecto al sistema para la planeación del manejo integral de los recursos forestales (SIPLAMIF) visto en el capítulo IV, se eligió para mostrar la aplicación real del modelo de objetos, en particular de la metodología de análisis y diseño orientado a objetos propuesta en éste trabajo. Cabe mencionar también que para comprender el problema, tuve la necesidad de involucrarme en el área del manejo de recursos forestales, lo cual implicó una serie de contratiempos, para poder concluir el presente trabajo.

Se puede concluir que los sistemas de manejo forestal, son sistemas que tienen una complejidad inherente. Deben manejar grandes volúmenes de información de origen variado: encuestas socioeconómicas y de precios; inventarios con información general de sitios y rodales, así como información de árboles; información obtenida de mapas y fotografías aéreas. Por otro lado deben manejar el conocimiento de expertos en manejo forestal, silvicultura, ecología, etc. Además existen relaciones complejas entre los componentes del sistema.

Una conclusión importante con respecto al análisis y diseño de este tipo de sistemas se basa en el hecho de que para obtener los beneficios del paradigma orientado a objetos, se debe poner especial atención en la definición precisa de cada una de las clases, así como en las relaciones que deben existir entre ellas. Si no se pone especial atención en la definición de las clases puede existir una reorganización masiva al intentarse alguna extensión o modificación al sistema.

En el diseño propuesto para el SIPLAMIF se localizan cuestiones de interés que es difícil manejar con otro paradigma de diseño (paradigma clásico). Por ejemplo, la definición de diferentes ecuaciones para cada uno de los

modelos dasométricos, aplicable a diferentes condiciones de bosques, debe realizarse en el lugar y momento preciso. Otro ejemplo lo constituye la complejidad de la relaciones que existen entre las clases obtenidas para el SIPLAMIF.

Con respecto al análisis podemos concluir que los métodos orientados a objetos posibilitan al equipo de desarrollo tener una mejor comprensión del dominio del problema que se pretende automatizar. Ya que como vimos en el análisis orientado a objetos, se busca modelar el mundo identificando las clases y objetos que conforman el vocabulario del dominio del problema.

Los analistas con más experiencia, realizan un análisis de dominios antes de implantar un sistema nuevo, ésta es una buena medida porque permite aprovechar la experiencia de otros proyectos similares, aprovechando de esta manera una de las ventajas que proporciona la orientación a objetos: la reutilización ya que solamente se tendrían que adaptar los marcos de referencia existentes, obteniendo además ahorros significativos en el costo de desarrollo.

Con respecto al diseño orientado a objetos, se puede concluir que nos permite crear arquitecturas sencillas entendibles y modificables.

Es importante recalcar que no se debe caer en los extremos de hacer diseños prematuros, es decir, antes de que termine la fase de análisis, porque es casi seguro que todavía no conocemos ni sabemos lo suficiente acerca del problema para hacer un buen diseño. Ni tampoco debemos empezar un diseño demasiado tarde porque caeríamos en el riesgo de desperdiciar recursos al querer hacer un modelo de análisis demasiado detallado o perfecto, y por lo tanto casi inalcanzable.

APÉNDICE

HERRAMIENTAS CASE COMERCIALES

Para que se tenga una visión general de los productos CASE comerciales, (hasta 1995), se incluye en este apéndice una descripción a manera de ficha sobre cada herramienta. Se incluye nombre del producto, año de la primera versión, así como de la versión actual, principales módulos, plataformas, fases del ciclo de vida soportadas así como las técnicas y metodologías que soportan. Para hacer una revisión de las herramientas CASE más actualizadas el lector puede consultar en la siguiente dirección de Internet: http://wwwis.cs.utwente.nl

Esta información ha sido obtenida de: Piattini, Mario G. *et. al. Elementos y Herramientas en el Desarrollo de Sistemas de Información, una visión actual de la tecnología CASE.* Addison-Wesley Iberoamericana S.A., E.U.A. 1995.

NOMBRE DEL PRODUCTO: BACHMAN

FABRICANTE:
BACHMAN INFORMATION SYSTEMS,INC.
New England Executive Park
Burlington, MAO1803
ESTADOS UNIDOS

DISTRIBUIDOR:
Level data S.A
Paseo de la Castellana, 140, 10ºD
28046 MADRID.
Tel: 564 72 74

AÑO PRIMERA VERSIÓN: 1987

VERSIÓN ACTUAL: 4.15 / 4.20 (Otoño 1994).

DESCRIPCIÓN:
BACHMAN Information Systems fundada en 1983 por Charles Bachman, inventor del primer sistema de gestión de base de datos.
El objetivo de BACHMAN Information Systems es desarrollar y comercializar un conjunto integrado de productos software orientados a la creación y mantenimiento de sistemas de información.
Las soluciones que ofrece están basadas en la reingeniería, reutilizacón e integración de información y su implementación en diferentes bases de datos y plataformas, incluyendo el entorno cliente/servidor. Entre otras soluciones que ofrece, incluyen el diseño, mantenimiento y optimización de bases de datos críticas, modelado de procesos, metodologías de trabajo y herramientas para la construcción de aplicaciones en tecnologías tradicionales y en cliente/servidor.La arquitectura abierta con que están diseñados sus productos permite crear aplicaciones que integran diversas fuentes de información, modificarlas en función de los cambios tecnológicos y de la actividad de la empresa, y ejecutarlas en una gran variedad de plataformas.

PRINCIPALES MÓDULOS:

- *BACHMAN/ Analyst*
- *BACHMAN/ DBA*
- *BACHMAN/ Designer*
- *BACHMAN/ Ellipse*
- *BACHMAN/ NETRON*

PLATAFORMAS:
OS/2, WINDOWS, AIX, UNIX,MVS (Según módulo).

FASES DEL CICLO DE VIDA SOPORTADAS:
Todas.

TÉCNICAS Y METODOLOGIAS:
Reingeniería análisis orientado a objetos, modelado gráfico, E/R, DFD, FDD, PFS, GANE & SARSON, BACHMAN, etc.
RSDM & adaptable a muchas metodologías y técnicas.

LENGUAJES GENERADOS:
COBOL, CPS, APS TELON, etc.

OBSERVACIONES:
Las herramientas BACHMAN se destacan por su facilidad de uso, reutilización através de técnicas de reingeniería y análisis orientado a objetos, integración de datos procesos y lógica que elimina muchas tareas repetitivas.

NOMBRE DEL PRODUCTO: INTEGRATED SYSTEMS DEVELOPMENT (ISD)

FABRICANTE:
BULL. S.A.
Tour Bull
Cedex 74
92039 Paris La Defense
France

DISTRIBUIDOR:
Bull (España), S.A.
Paseo Doce Estrellas, 2
28042, Madrid.

AÑO PRIMERA VERSIÓN: 1993

VERSIÓN ACTUAL: V 1.2

DESCRIPCIÓN:
ISD proporciona a los desarrolladores una plataforma de servicios básada en dos modelos de referencia : el DCM (Distributed Computer Model) para conectividad de entornos de información heterogéneos y el modelo general para entornos de desarrollo PCTE.

Es un entorno de integración en el que una amplia variedad de herramientas de diferentes proveedores pueden trabajar juntas, compartir información, intercambiar peticiones y mensajes y presentarse de manera consistente al desarrollador, ofreciendole integración de presentación, datos y control.

ISD está compuesto por dos grupos de componentes: un grupo es una implementación de PCTE; el segundo grupo construido sobre PCTE, es una serie de productos de intercambio que traducen la salida de herramientas a una forma neutral, la almacena en un depósito y permite que sea compartida por otras herramientas.

PRINCIPALES MÓDULOS:
- *Integrated Desktop*
- *Integrated Exchange Server*
- *Integrated Exchange Options*
- *Integrated PCTE Server*
- *Integrated PCTE Options*
- *Integrated PCTE Graphical Tools*
- *Integrated Repository Tools*
- *Integrated Repository Models*

PLATAFORMAS:
UNIX, PCs

FASES DEL CICLO DE VIDA SOPORTADAS:
Todas

TÉCNICAS Y METODOLOGIAS:
Usa técnicas de bases de datos orientadas a objetos, que incluye servicios e interfases para asegurar su portabilidad. Es independiente de toda metodología.

OBSERVACIONES:
No es una herramienta, sino un integrador de herramientas CASE, aporta beneficios a usuarios de paquetes CASE integrados que presentan ciertas deficiencias en el saporte para algunas fases del ciclo de vida, a usuarios de componentes CASE de fase especifica diferentes, a constructores de herramientas CASE y a especialistas en integración de sistemas.

NOMBRE DEL PRODUCTO: APPLICATION DEVELOPMENT WORKBENCH (ADW)

FABRICANTE:
KnowledgeWare
Atlanta, USA.

DISTRIBUIDOR:
KnowledgeWare Int.
Alberto Alcocer, 24 piso 3
28036, Madrid Esp.

AÑO PRIMERA VERSIÓN: 1984

VERSIÓN ACTUAL: 2.7

DESCRIPCIÓN:
El ADW es una herramienta CASE gráfica integrada que funciona en PC. Asiste al usuario en todas las fases del ciclo de vida. El trabajo fluye de una manera eficiente desde la aplicación estratégica hasta el diseño detallado y la construcción.

Tda la información recogida durante el ciclo de vida del desarrollo es almacenada en una enciclopedia central, la cual es el punto de integración de todos los productos de la línea ADW.

La enciclopedia asegura la consistencia de las especificaciones y diseño, y permite la verificación de errores en tiempo real para garantizar la calidad del proceso.

Debido a que ADW ayuda al análisis sistemático de los requerimientos, las aplicaciones desarrolladas cubren los requeerimientos de la empresa desde el momento que son implantadas. La facilidad para entender los diagramas, tablas y matrices aumenta la comunicación entre todas las personas involucradas en el proceso de desarrollo. Además ya que todos los rerquerimientos han sido capturados en la enciclopedia, no existe el riesgo de perderlos u olvidarlos. Los datos y procesos lógicos almacenados pueden ser reutilizados por otros eqipos del proyecto, reduciendo el tiempo de desarrollo y asegurando la consistencia de los mismos a través de la organización.

Con ADW, se analizan y definen los requerimientos sin tomar en cuenta la plataforma tecnológica de la empresa . Posteriormente se pueden desarrollar las especificaciones de diseño y la generación de código para diferentes entornos de hardware.

PRINCIPALES MÓDULOS:

- *ADW /Planning*
- *ADW/RAD*
- *ADW/Construction*
- *ADW/Analisys*
- *ADW/Design*
- *ADW/Doc*

PLATAFORMAS:
PC con OS/2

FASES DEL CICLO DE VIDA SOPORTADAS:
Planeación estratégica, análisis, diseño, construcción y mantenimiento

TÉCNICAS Y METODOLOGIAS:
ADW es una herramienta abierta y no está condicionada con ninguna metodología en especial.
Las tecnicas sosportadas son las siguientes:
E/R, DFD, diagramas de descomposición, matrices (asociación, propiedad, seguimiento), control de normalización hasta la forma normal 1,2 o 3, diagrama de estructuras, prototipos.

OBSERVACIONES:
Recientemente se liberó la versión3.X de ADW. A corto plazo se podrá disponer del ADW en el entorno Windows NT y UNIX:

NOMBRE DEL PRODUCTO: OBSYDIAN

FABRICANTE:
Synon Inc.
1100 Larkspur Landing Circle
Suite 300
Larkspur, CA. 94939

DISTRIBUIDOR:
Consulting Informático, S.A. - (CINSA)
Santa María Magdalena, 15
28016, Madrid

AÑO PRIMERA VERSIÓN: 1994

VERSIÓN ACTUAL: 1.0

DESCRIPCIÓN:
Obsydian es un entorno de aplicaciones que acelera la implantación de sistemas empresariales a nivel global, mediante su capacidad de definir y reulizar objetos abstractos de negocios.
Obsydian combina la abtracción de los modelos empresariales de la ingeniería de la información con los principios de reutilización de software orientado a objetos para mejorar la calidad y elevar la productividad del desarrollo del software.
Obsydian da soporte a entornos de desarrollo de aplicaciones complejos y de alto rendimiento, facilitando:
- Capacidad de diseño y construcción totalmente integradas.
- Reutilización de objetos de negocios para acelerar la entrega de aplicaciones.
- Herramientas que facilitan el trabajo en grupo en grandes desarrollos.

Obsydian esta diseñado para crear aplicaciones que soporten:
- Nuevos modelos de empresa basados en arquitecturas cliente/servidor, host y computadoras personales.
- Demandas masivas de datos y transacciones.
- Los mayores niveles de integridad, control y seguridad.
- Ejecución y optimización en múltiples plataformas.

Entre otras carácteristicas Obsydian permite desarrollar aplicaciones críticas para la empresa, adaptándose a las condiciones cambiantes del negocio y explotando completamente las nuevas tecnologías.

PRINCIPALES MÓDULOS:
Obsydian Módulo de Diseño: Incluye el modelador de datos, de procesos, diccionario de grupos, administrador de versiones, ojeador de objetos y editor de GUI.
Obsidian Biblioteca de Clases: Incluye objetos de diseño básico y objetos de negocios abstractos y básicos, es decir, objetos con sus atributos, estructuras, comportamientos y componentes de diseño e implantación reutilizables para aplicaciones empresariales comunes.
Obsydian Generador de Entorno: Todos los generadores incluyen módulo de acceso a la base de datos, lengiajes, interfaces y generadores ed ayuda para:

- Generador clientes servidor Microsoft Windows AS/400 (C++, RPG/400)
- Generador clientes servidor Microsoft Windows HP/9000 (C++, C)
- Generador clientes servidor Microsoft Windows RS/6000 (C++, C)
- Generador para AS/400 NPT 5250. (RPG/400)

PLATAFORMAS:
Microsoft Windows 3.1 o superior

FASES DEL CICLO DE VIDA SOPORTADAS:
Análisis, diseño, construcción y mantenimiento.

TÉCNICAS Y METODOLOGIAS:
Orientación a objetos
Ingeniería de la información

LENGUAJES GENERADOS:
C++, C, RPG.
DB2/400, Oracle, Sybase, Informix, ODBC, EDA/SQL.

NOMBRE DEL PRODUCTO: OMW (OBJECT MANAGEMENT WORKBENCH)

FABRICANTE:
Intellicorp
1975 El camino Real West
Montain View, CA. 94040-2216

DISTRIBUIDOR:
SLIGOS
SERVICIOS INFORMATICOS
Paseo de la Castellana 95

AÑO PRIMERA VERSIÓN: 1993

DESCRIPCIÓN:
OMW implementa la Metodología de Martin /Odell Object Oriented Information Engineering (OOIE) y tiene una original visión de CASE: modela el negocio que ejecuta.
OMW es una clase de entorno de desarrollo visual que da soporte a todo el ciclo de vida desde el análisis y el diseño hasta la entrega del producto final.
Las herramientas gráficas de OMW modelan los problemas del negocio y así los usuarios, analistas y desarrolladores pueden contribuir a construir el diseño.
Los modelos son completamente ejecutables y testeables y se pueden usar directamente como la base para el desarrollo de aplicaciones.
Las herramientas de OMW son totalmente Object- Oriented y permiten usar la variedad de técnicas de modelado como son:
- *Object Diagramer*, permite modelar los objetos y las asociaciones entre ellos.
- *Event Diagramer*,Presenta la información del flujo de los procesos. Muestra relaciones entre los procedimientos del negocio, operaciones e interfases gráficas de usuario.

Al ser modelos ejecutables, existe una animación gráfica que proporciona unflujo visual del proceso.
- *Bussines Rule Editor*, convierte las reglas del negocio en explicitas, visibles y fáciles de modificar y validar
- *Scenario Manager*, permite ejecutar y validar los modelos graficos del negocio creados en OMW. Se pueden analizar test como situaciones de la vida real.
- Documentación e informes auutomáticos para aplicaciones OMW.

PLATAFORMAS:
UNIX

FASES DEL CICLO DE VIDA SOPORTADAS:
Análisis, diseño, construcción prueba , entrega.

LENGUAJES GENERADOS:
C, PROTALK (Parecido a SMALLTALK)

OBSERVACIONES:
Por tratarse de una metodología de orientación a objetos, proporciona soluciones reutilizables y mantenibles. Es muy eficiente en cuanto al modelado de los eventos.

NOMBRE DEL PRODUCTO: ORACLE CASE

FABRICANTE:
Oracle, Corp.
500 Oracle Parkway
Redwood City, CA.
94065, USA.

DISTRIBUIDOR:
Oracle Ibérica S.A.
Urb. La Florida
Gobelas 33

AÑO PRIMERA VERSIÓN: 1987

VERSIÓN ACTUAL: 5.1

DESCRIPCIÓN:
Oracle Case es un conjunto de herramientas integradas que cubren todo el ciclo de vida de los sistemas de información. Cuenta con una metodología propia, denominada Case Method, que desglosa el ciclo de vida en las fases de estrategia, análisis diseño e implementación.

PRINCIPALES MÓDULOS:
Oracle Dictionary Interfase del repositorio activo, multiusuario, multiversión, extensible.
Oracle Designer Harramienta gráfica que soporta diagramas entidad-relación, DFD, descomposición funcional y diagramas de matriz.
Oracle Generator Generadores de programas interactivos e informes basados en Oracle Forms y Oracle Reports.

PLATAFORMAS:
Cliente/Servidor. La práctica totalidad de plataformas UNIX así como VMS, MVS, Netware, Windows NT, OS/2, etc. pueden comportarse como servidores RDBMS Oracle.
Oracle Case puede ser un cliente en: MS-Windows, Unix/Motif, VMS/Motif, OS/2

FASES DEL CICLO DE VIDA SOPORTADAS:
Todas las fases del ciclo de vida

TÉCNICAS Y METODOLOGIAS:
Oracle Case incorpora diagramas entidad-relación para el modelado de datos y descomposición de funciones y diagramas de flujo de datos para el modelado de procesos.
A la vez, dispone de diagramas matriciales abiertos para obtener e incorporar información cruzada sobre cualquier clase de objeto del repositorio.

LENGUAJES GENERADOS:

Oracle Forms	Oracle Reports
SQL Forms	SQL ReportWriter
SQL Plus	PL SQL
SQL TextRetrieval	

NOMBRE DEL PRODUCTO: StP/OMT (Object Modeling Technique)

FABRICANTE:	**DISTRIBUIDOR:**
IDE Interactive Development Environments San Francisco California EE.UU.	Tecnova Consultores e Ingenieros de Sistemas de Información Velázquez 94, 4 Izq. 28006, Madrid.

AÑO PRIMERA VERSIÓN: 1993 **VERSIÓN ACTUAL:** 2.0

DESCRIPCIÓN:
StP. OMT es un entorno de desarrollo integrado multiusuario que da soporte completo a la metodología orientada a objetos, OMT (Object Modeling Technique) de Rumbaugh *et al.*
StP/OMT permite navegar entre los diferentes editores permitiendo er completando las especificaciones del sistema con metodología orientada a objetos. Se puede ir pasando de editor en editor que dan soporte a los tres modelos que conforman la metodología: modelo de objetos, modelo dinámico y modelo funcional, e ir definiendo cada uno de los componenetes del sistema. Asociados a los editores, existen, tablas complementarias que se pueden ir rellenando para completar la información para cada modelo(definición de clase, definición de estados, etc.).
Dentro de la metodología, también da soporte a la extensión para tiempo real. Por último , cabe destacar que posee un módulo de captura de clases C++, lo que permite realizar tareas de ingeniería inversa desde código fuente.
En cuanto a la generación de código, existen entornos específicos para generación C++, Ada, Smalltalk e IDL.

PRINCIPALES MÓDULOS:
- Editor del modelo de objetos
- Editor del modelo dinámico
- Editor de tabla de clases
- Browser de clases
- Librería de reutilización de clases
- Generador de clases OMT a partir de código C++ existente
- Activación de Scripts de QRL
- Editor del modelo funcional
- Editor de escenarios
- Editor de tabla de estados
- Browser de subsistemas
- Generador incremental de código C++
- Gestión de impresión
- Gestión de versiones

PLATAFORMAS:

Sun Sparc	Sun OS y Solaris
HP 700/800	HP-UX
DEC Alpha	OSF/1
RS/6000	AIX

FASES DEL CICLO DE VIDA SOPORTADAS:
- Análisis
- Diseño
- Implementación
- Ingeniería Inversa

TÉCNICAS Y METODOLOGIAS:
OMT (Object Modelling Technique, Rumbaugh, *et.al.*)

LENGUAJES GENERADOS:
C++, Smalltalk, Ada, SQL e IDL

OBSERVACIONES:
Dos caracteristicas a destacar de este producto:
1. ***Adaptabilidad***- El usuario puede personalizar todos los aspectos de los editores y el entorno de trabajo mediante un lenguaje común de definición, permite modificar la representación de los objetos.
2. ***Actualización***- Debido a la estrecha colaboración entre IDE, Rumbaugh y ACC, StP/OMT estará siempre por delante con nuevas versiones.

GLOSARIO

abstracción. Las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporcionan así fronteras conceptuales definidas con nitidez en relación con la perspectiva del observador; el proceso de centrarse en las características esenciales de un objeto. La abstracción es uno de los elementos fundamentales del modelo de objetos.

abstracción clase. Clase u objeto que forma parte del vocabulario del dominio del problema.

acción. Una operación que, a efectos prácticos, precisa de un tiempo cero para realizarse. Una acción puede denotar la invocación de un método, el disparo de otro evento, o el lanzamiento o parada de una actividad.

actividad. Una operación que precisa algún tiempo para completarse.

actor. Objeto que puede operar sobre otros objetos pero sobre el que nunca operan otros objetos. En algunos contextos, los términos *objeto activo* y *actor* son equivalentes.

agente. Objeto que puede operar sobre otros objetos y sobre el que pueden operar otros objetos. Un agente suele crearse para realizar algún trabajo en nombre de un actor u otro agente.

análisis orientado a objetos. Método de análisis en el que los requisitos se examinan desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema.

arquitectura. La estructura lógica y física de un sistema, forjada por todas las decisiones de diseño estratégicas y tácticas aplicadas durante el desarrollo.

arquitectura de módulos. Gráfico cuyos vértices representan módulos y cuyos arcos representan relaciones entre esos módulos. La arquitectura de módulos de un sistema se representa por un conjunto de diagramas de módulos.

arquitectura de procesos. Gráfico cuyas vértices representan procesadores y dispositivos y cuyos arcos representan conexiones entre estos procesadores y dispositivos. La arquitectura de procesos de un sistema se representa mediante un conjunto de diagramas de procesos.

asociación. Relación que denota una conexión semántica entre dos clases.

atributo. Una parte de un objeto agregado.

campo. Depósito para parte del estado de un objeto; colectivamente, los campos de un objeto constituyen su estructura. Los términos *campo*, *variable de instancia*, *objeto miembro* y *ranura o slot* son intercambiables.

capa. Colección de categorías de clases o subsistemas al mismo nivel de abstracción.

cardinalidad. El número de instancias que puede tener una clase; el número de instancias que participan en una relación entre clases.

categoría de clases. Colección lógica de clases, algunas de las cuales son visibles para otras categorías de clases, y otras de las cuales están ocultas. Las clases de una categoría de clases colaboran para proporcionar un conjunto de servicios.

clase. Conjunto de objetos que comparten una estructura común y un comportamiento común. Los términos *clase* y *tipo* suelen ser (no siempre) equivalentes; una clase es un concepto ligeramente diferente del de un tipo, en el sentido de que enfatiza la clasificación de estructura y comportamiento.

clase abstracta. Una clase que no tiene instancias. Una clase abstracta se escribe con la intención de que sus subclases concretas añadan elementos nuevos a su estructura y comportamiento, normalmente implantando sus operaciones abstractas.

clase base. La clase más generalizada en una estructura de clases. La mayoría de las aplicaciones tiene muchas de tales clases raíz. Algunos lenguajes definen una clase base primitiva, que sirve como la superclase última de todas las clases.

clase contenedor. (container) Clase cuyas instancias son colecciones de otros objetos. Las clases contenedor pueden denotar colecciones homogéneas (todos los objetos de la colección son de la misma clase) o heterogéneas (cada uno de los objetos de la colección puede ser de una clase diferente, aunque generalmente todos deben compartir una superclase común). Las clases contenedor se definen la mayoría de las veces como clases parametrizadas, en las que algún parámetro designa la clase de los objetos contenidos.

clase genérica. Clase que sirve como plantilla para otras clases, en las que la plantilla puede parametrizarse con otras clases, objetos y/u operaciones. Una clase genérica debe ser instanciada (rellenados sus parámetros) antes de que puedan crearse objetos. Las clases genéricas se usan típicamente como clases contenedor. Los términos *clase genérica* y *clase parametrizada* son intercambiables.

clave. Atributo cuyo valor identifica de forma única un solo objeto.

cliente. Objeto que usa los servicios de otro objeto, ya sea operando sobre él o haciendo referencia a su estado.

colaboración. El proceso por el cual varios objetos cooperan para proporcionar algún comportamiento de nivel superior.

complejidad espacial. Tiempo absoluto o relativo en el que se completa alguna operación.

comportamiento. Cómo actúa y reacciona un objeto, en términos de sus cambios de estado y su paso de mensajes; la actividad exteriormente visible y comprobable de un objeto.

comprobación estricta de tipos. Una característica de un lenguaje de programación, de acuerdo con la cual se garantiza que todas las expresiones son consistentes respecto al tipo.

comprobación de tipos (typing). El hacer cumplir la clase de un objeto, lo que previene el intercambio de objetos de diferentes tipos o, a lo mucho, permite ese intercambio sólo de maneras muy restringidas.

concurrencia. Propiedad que distingue un objeto activo de uno que no es activo.

control de acceso. El mecanismo de control de acceso a la estructura o comportamiento de una clase. Los elementos públicos son accesibles por todo el mundo; los protegidos (protected) son accesibles sólo por las subclases, implantación y amigos de la clase en cuestión; los elementos privados son accesibles sólo por la implantación y amigos de la clase que contiene el elemento; los elementos de implantación son accesibles sólo por la implantación de la clase que contiene el elemento.

decisión de diseño estratégica. Una decisión de diseño que tiene vastas implicaciones arquitectónicas.

decisión de diseño táctica. Decisión de diseño que tiene implicaciones arquitectónicas locales.

delegación. El acto por el cual un objeto transmite una operación a otro objeto, para que este la realice en nombre del primero.

descomposición algorítmica. El proceso de dividir un sistema en partes, cada una de las cuales representa algún paso pequeño de un proceso más grande. La aplicación de métodos de diseño estructurado conduce a una descomposición algorítmica, cuyo centro de interés está en el flujo de control dentro de un sistema.

descomposición orientada a objetos. El proceso de dividir un sistema en partes, cada una de las cuales representa alguna clase u objeto del dominio del problema. La aplicación de métodos de diseño orientado a objetos lleva a una descomposición orientada a objetos, en la que se ve el mundo como una colección de objetos que cooperan con otros para conseguir alguna funcionalidad que se desea.

destructor. Operación que libera el estado de un objeto y/o destruye el propio objeto.

diagrama de clases. Parte de la notación del diseño orientado a objetos, utilizada para mostrar la existencia de las clases y sus relaciones en el diseño lógico de un sistema. Un diagrama de clases puede representar todo o parte de la estructura de clases de un sistema.

diagrama de interacción. Parte de la notación del diseño orientado a objetos, utilizada para mostrar la ejecución de un escenario en el contexto de un diagrama de objetos.

diagrama de módulos. Parte de la notación del diseño orientado a objetos, utilizada para mostrar la asignación de clases y objetos a módulos en el diseño físico de un sistema.

Un diagrama de módulos puede representar toda la arquitectura de módulos de un sistema o solamente una parte de ella.

diagrama de objetos. Parte de la notación del diseño orientado a objetos, utilizada para mostrar la existencia de objetos y sus relaciones en el diseño lógico de un sistema. Un diagrama de objetos puede representar toda la estructura de objetos de un sistema o parte de ella, e ilustra principalmente la semántica de los mecanismos del diseño lógico. Un solo diagrama de objetos representa una instantánea de lo que de otro modo es un evento o configuración transitoria de los objetos.

diagrama de procesos. Parte de la notación del diseño orientado a objetos, utilizada para mostrar la asignación de procesos a procesadores en el diseño físico de un sistema. Un diagrama de procesos puede representar todo o parte de la arquitectura de procesos de un sistema.

diagrama de transición de estados. Parte de la notación del diseño orientado a objetos, utilizada para mostrar el espacio de 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.

diccionario de datos. Depósito amplio que enumera todas las clases de un sistema.

diseño estructurado. Método de diseño que abarca el proceso de descomposición algorítmica.

diseño global circular. (round-trip gestalt design) Un estilo de diseño que enfatiza el desarrollo incremental e iterativo de un sistema, mediante el refinamiento de vistas lógicas (diferentes pero consistentes del sistema como un todo; el proceso de diseño orientado a objetos está guiado por los conceptos del diseño global circular; el diseño global circular es un reconocimiento del hecho de que la visión global de un diseño influye en sus detalles, y los detalles afectan con frecuencia a la visión global.

diseño orientado a objetos. método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir modelos lógicos y físicos, así como estáticos y dinámicos, del sistema que se diseña; en concreto, esta notación incluye diagramas de clases, diagramas de objetos, diagramas de módulos y diagramas de procesos.

dispositivo. Elemento de hardware que no tiene recursos computacionales.

encapsulamiento. El proceso de introducir en el mismo compartimiento los elementos de una abstracción que constituyen su estructura y comportamiento; el encapsulamiento sirve para separar el interfaz contractual de una abstracción y su implantación.

enlace. Entré dos objetos, una instancia de una asociación.

escenario. Esquema de los eventos que provocan algún comportamiento en el sistema.

espacio de estados. Enumeración de todos los estados posibles de un objeto. El espacio de estados de un objeto abarca un número indefinido pero finito de estados posibles (aunque no siempre deseables o esperados).

estado. Resultados acumulados del comportamiento de un objeto; una de las condiciones posibles en que puede existir un objeto, caracterizada por cantidades definidas que

son distintas de otras; en cualquier momento dado, el estado de un objeto abarca todas las propiedades (normalmente estáticas) del objeto más los valores actuales (normalmente dinámicos) de cada una de esas propiedades.

estructura. Representación concreta del estado de un objeto. Un objeto no comparte su estado con ningún otro objeto, aunque todos los objetos de la misma clase comparten la misma representación de su estado.

estructura de clases. Gráfico cuyos vértices representan clases y cuyos arcos representan relaciones entre esas clases. La estructura de clases de un sistema se representa mediante un conjunto de diagramas de clases.

estructura de objetos. Gráfico cuyos vértices representan objetos y cuyos arcos representan relaciones entre esos objetos. La estructura de objetos de un sistema se representa mediante un conjunto de diagramas de objetos.

evento. Algún suceso que puede hacer que cambie el estado de un sistema.

fichas CRC Clase/Responsabilidades/Colaboradores; una herramienta simple para efectuar tormentas de ideas sobre las abstracciones y mecanismos clave de un sistema.

función. Correspondencia entrada/salida que resulta del comportamiento de algún objeto.

función genérica Una operación sobre un objeto. Una función genérica de una clase puede redefinirse en las subclases; así, para un objeto dado, se implementa mediante un conjunto de métodos declarados en varias clases relacionadas mediante su jerarquía de herencias. Los términos *función genérica* y *función virtual* suelen ser equivalentes.

función miembro. Operación sobre un objeto, definida cómo parte de la declaración de una clase; todas las funciones miembro son operaciones, pero no todas las operaciones son funciones miembro. Los términos *función miembro* y *método* suelen ser equivalentes. En algunos lenguajes, las funciones miembro son independientes y pueden redefinirse en una subclase; en otros lenguajes, las funciones miembro no pueden redefinirse, pero sirven como parte de la implementación de una función genérica o virtual, las cuales pueden redefinirse ambas en una subclase.

función virtual. Operación sobre un objeto. Una función virtual puede ser redefinida por las subclases; así, para un objeto dado, se implementa mediante un conjunto de métodos declarados en diversas clases que se relacionan mediante su jerarquía de herencias. Los términos *función genérica* y *función virtual* suelen ser intercambiables.

herencia. Relación entre clases, en la que una clase comparte la estructura o comportamiento definido en otra (herencia simple) u otras (herencia múltiple) clases. La herencia define una relación de tipo entre clases en la que una subclase hereda de una o más superclases generalizadas; una subclase suele especializar a sus superclases aumentando o redefiniendo la estructura y comportamiento existentes.

hilo de control. Un solo proceso. El comienzo de un hilo de control es la raíz a partir de la cual ocurren las acciones dinámicas independientes dentro de un sistema; un sistema dado puede tener muchas hilos simultáneos de control, algunos de los cuales pueden aparecer dinámicamente y dejar después de existir. Los sistemas que se ejecutan en múltiples CPUs permiten hilos de control verdaderamente concurrentes, mientras que

los sistemas que corren en una sola CPU sólo pueden conseguir la ilusión de hilos concurrentes de control.

identidad. La naturaleza de un objeto que lo distingue de todos los demás.

implementación. Vista interna de una clase, objeto o módulo, que incluye los secretos de su comportamiento.

instancia. Algo a lo cual se le pueden hacer cosas. Una instancia tiene estado, comportamiento e identidad. La estructura y comportamiento de instancias similares se definen en su clase común. Los términos *instancia* y *objeto* son intercambiables.

interfaz. Vista externa de una clase, objeto o módulo, que enfatiza su abstracción mientras que oculta su estructura y los secretos de su comportamiento.

ingeniería directa. Producción de código ejecutable a partir de un modelo físico o lógico.

ingeniería inversa. Producción de un modelo lógico o físico a partir de código ejecutable.

instanciación. Proceso de rellenar la plantilla de una clase genérica o parametrizada para producir una clase a partir de la cual pueden crearse instancias.

iterador. Operación que permite visitar las partes de un objeto.

jerarquía. Clasificación u ordenación de abstracciones. Las dos jerarquías más habituales en un sistema complejo son su estructura de clases y su estructura de objetos; pueden encontrarse también jerarquías en las arquitecturas de módulos y procesos de un sistema complejo.

ligadura dinámica. Ligadura denota la asociación de un nombre (como una declaración de variable) con una clase; ligadura dinámica es una ligadura en la que la asociación nombre/ clase no se realiza hasta que el objeto designado por el nombre se crea en tiempo de ejecución.

ligadura estática. Ligadura denota la asociación de un nombre (como una declaración de variable) con una clase; ligadura estática es una ligadura en la que la asociación nombre/ clase se realiza cuando se declara el nombre (en tiempo de compilación) pero antes de la creación del objeto que designa el nombre.

marco de referencia. Colección de clases que proporcionan un conjunto de servicios para un dominio particular; un marco de referencia exporta por tanto una serie de clases y mecanismos individuales que los clientes pueden usar o adaptar.

mecanismo. Estructura por la que los objetos colaboran para proporcionar algún comportamiento que satisface un requisito del problema.

mensaje. Operación que un objeto realiza sobre otra. Los términos *mensaje*, *método* y *operación* suelen ser equivalentes.

metaclase. La clase de una clase; una clase cuyas instancias son a su vez clases.

método. Operación sobre un objeto, definida como parte de la declaración de una clase; todos los métodos son operaciones, pero no todas las operaciones son métodos. Los términos *mensaje*, *método* y *operación* suelen ser equivalentes. En algunos lenguajes, los

métodos son independientes y pueden redefinirse en una subclase, en otros, los métodos no pueden redefinirse, pero sirven como parte de la implementación de una función genérica o virtual, que pueden redefinirse ambas en una subclase.

modelo de objetos. La colección de principios que forman las bases del diseño orientado a objetos; un paradigma de ingeniería del software que enfatiza los principios de abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia.

modularidad. Propiedad de un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.

módulo. Unidad de código que sirve como bloque de construcción para la estructura física de un sistema; una unidad de programa que contiene declaraciones, expresadas en el vocabulario de un lenguaje de programación concreto, que forma la realización física de algunas o de todas las clases y objetos del diseño lógico del sistema. Un módulo suele tener dos partes: su interfaz y su implementación.

nivel de abstracción. Clasificación relativa de las abstracciones en una estructura de clases, estructura de objetos, arquitectura de módulos o arquitectura de procesos. En términos de su jerarquía «de partes», una abstracción dada está a un nivel de abstracción más alto que otras si se construye sobre las otras; en términos de su jerarquía «de tipos», las abstracciones de alto nivel están generalizadas, y las de bajo nivel están especializadas.

objeto. Algo a lo cual se le pueden hacer cosas. Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares se definen en su clase común. Los términos *instancia* y *objeto* son intercambiables.

objeto activo Objeto que contiene su propio hilo de control.

objeto agregado. Objeto compuesto de otro u otros objetos, cada uno de los cuales se considera parte del objeto agregado.

objeto concurrente. Objeto activo cuya semántica se garantiza en presencia de múltiples hilos de control.

objeto miembro. Repositorio para parte del estado de un objeto; colectivamente, los objetos miembro de un objeto constituyen su estructura. Los términos *campo*, *variable de instancia*, *objeto miembro* y *ranura* (slot) son intercambiables.

objeto pasivo. Un objeto que no contiene su propio hilo de control.

ocultación de información. Proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto está oculta, así como la implantación de sus métodos.

operación. Algún trabajo que un objeto realiza sobre otro con el fin de provocar una reacción. Todas las operaciones sobre un objeto concreto pueden encontrarse en forma de subprogramas libres y funciones miembro o métodos. Los términos *mensaje*, *método* y *operación* suelen ser equivalentes.

operación abstracta. Operación que es declarada pero no implantada por una clase abstracta. En C++, una operación abstracta se declara como una función miembro virtual pura.

operación de clases. Operación, como por ejemplo un constructor o destructor, dirigida a una clase en vez de a un objeto.

papel (rol). El propósito o capacidad por el que una clase u objeto participa en una relación con otro; el papel de un objeto denota la selección de un conjunto de comportamientos bien definidos en un solo punto del tiempo; un papel es la cara que un objeto presenta al mundo en un momento dado.

partición. Las categorías de clases o subsistemas que forman parte de un nivel dado de abstracción.

persistencia. La propiedad de un objeto por la 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 (la ubicación del objeto se mueve del espacio de direcciones en el que se creó).

polimorfismo. Un concepto de teoría de tipos, de acuerdo con el cual un nombre (como una declaración de variable) puede denotar objetos de muchas clases diferentes que se relacionan mediante alguna superclase común; así, todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes modos.

privada. (private) Declaración que forma parte del interfaz de una clase, objeto o módulo; lo que se declara como privado (private) no es visible para ninguna otra clase, objeto o módulo.

proceso. Activación de un solo hilo de control.

procesador. Elemento de hardware que tiene recursos computacionales.

programación basada en objetos. Método de programación, en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de algún tipo, y cuyos tipos son miembros de una jerarquía de tipos unidos mediante relaciones que no son de herencia. En tales programas, los tipos suelen verse como estáticos, mientras que los objetos suelen tener una naturaleza mucho más dinámica, un tanto restringida por la existencia de la ligadura estática.

programación orientada a objetos. Método de implantación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son miembros de una jerarquía de clases unidas mediante relaciones de herencia. En tales programas, las clases suelen verse como estáticas, mientras que los objetos suelen tener una naturaleza mucho más dinámica, promovida por la existencia de la ligadura dinámica y el polimorfismo.

protegida (protected). Declaración que forma parte del interfaz de una clase, objeto o módulo, pero que no es visible para ninguna otra clase, objeto o módulo excepto los que representan subclases.

protocolo. Las formas de las que un objeto puede actuar y reaccionar, constituyendo la vista estática y dinámica externas completas del objeto; el protocolo de un objeto define la parte externa del comportamiento que se permite a un objeto.

pública (public). Declaración que forma parte del interfaz de una clase, objeto o módulo, que es visible para todas las demás clases, objetos y módulos que tienen visibilidad para él.

punto funcional. En el contexto de un análisis de requisitos, una actividad simple, visible y comprobable exteriormente.

ranura (slot). Depósito para parte del estado de un objeto; colectivamente, las ranuras de un objeto constituyen su estructura. Los términos *campo*, *variable de instancia*, *objeto miembro* y *ranura* son intercambiables.

responsabilidad. Algún comportamiento para el cual se cuenta con un objeto; una responsabilidad denota la obligación de un objeto para proporcionar cierto comportamiento.

restricción. Expresión de alguna condición semántica que debe protegerse.

selector. Operación que accede al estado de un objeto pero no lo altera.

servicio. Comportamiento proporcionado por una parte dada de un sistema.

servidor. Objeto que nunca opera sobre otros, sino que sólo se opera sobre él por parte de otros objetos; un objeto que proporciona ciertos servicios.

signatura (signature). Perfil completo de los argumentos formales y tipo de retorno de una operación.

sincronización. La semántica de concurrencia de una operación. Una operación puede ser simple (sólo conlleva un hilo de control), síncrona (cita entre dos procesos), de abandono inmediato (un proceso puede citarse con otro sólo si el segundo proceso ya está esperando), de tiempo limitado (un proceso puede citarse con otro, pero esperará por el segundo sólo durante un tiempo determinado), o asíncrono (los dos procesos operan independientemente).

sistema en tiempo real. Sistema cuyos procesos esenciales deben satisfacer ciertas restricciones criticas de tiempo. Un sistema en tiempo real riguroso debe ser determinista; el perder una de esas restricciones puede traer consecuencias catastróficas.

subclase. Clase que hereda de una o más clases (llamadas sus superclases inmediatas).

subsistema. Colección de módulos, algunos de los cuales son visibles a otros subsistemas y otros de los cuales están ocultos.

superclase. La clase de la cual hereda otra clase (llamada su subclase inmediata).

tipo. Definición del dominio de valores admisibles que un objeto puede tener y del conjunto de operaciones que pueden realizarse sobre ese objeto. Los términos *clase* y *tipo* suelen ser (no siempre) equivalentes; un tipo es un concepto ligeramente diferente de una clase, en el sentido de que enfatiza la importancia de la conformidad con un protocolo común.

transición. El paso de un estado a otro estado.

utilidad de clase. Colección de subprogramas libres o, en C++, una clase que sólo proporciona miembros static y/o funciones miembro static.

variable de clase. Parte del estado de una clase. Colectivamente, las variables de clase de una clase constituyen su estructura. Una variable de clase es compartida por todas las instancias de la misma clase. En C++, una variable de clase se declara como un miembro static.

variable de instancia. Depósito para parte del estado de un objeto. Colectivamente, las variables de instancia de un objeto constituyen su estructura. Los términos *campo*, *variable de instancia*, *objeto miembro* y *ranura o slot* son intercambiables.

BIBLIOGRAFÍA

- Booch, Grady. *Object Oriented Design with aplications*, Benjammin Cummings Publishing, Co. U.S.A. 1994.

- Booch, Grady. *Análisis y Diseño Orientado a Objetos con aplicaciones.* segunda edición, Addison Wesley, México, 1996.

- Burch, John G. *Diseño de sistemas de información, teoría y práctica*, Limusa, México, 1992.

- Cárdenas García, Sergio R. artículo: *Una panorámica de la ingeniería de software* en la revista Soluciones Avanzadas, núm. 8, marzo-abril, México, 1994.

- Diario Oficial de la Federación del 30 de marzo de 1986.

- Fernández, Villaseñor Yolanda. *Servicios de Manejo del conocimiento en ambientes de apoyo a la investigación científica*, IV Simposium Internacional sobre Inteligencia Artificial, Cancún, México Nov. 13-15 de 1991.

- Gerez, Victor. *et. al. Introducción al análisis de sistemas e investigación de operaciones*, Representaciones y Servicios de Ingeniería, México, 1978.

- Graham, Ian. *Métodos Orientados a Objetos.* segunda edición, Adisson Wesley Diaz de Santos. E.U.A. 1996. pp.

- Lawless, Jo A. *et. al. Undestanding CLOS: the Common Lisp Object System*, digital press, U.S.A. 1991.

- Lindsay, Susan. *Practical aplicatios of expert systems*, QED information sciences, U.S.A. 1988.

- Martin, James. *et. al. Análisis y Diseño Orientado a Objetos.* Prentice Hall Hispanoamericana, México. 1994.

- McClure, Carma. *CASE la automatización del software,* ed. Addison Wesley Iberoamericana, S.A., E.U.A. 1993.

- Oktaba, Hanna. artículo: El impacto de la ingeniería de software en la programación orientada a objetos, México, UNAM, 1991.

- Padilla, G. H. *Glosario práctico de términos forestales.* LIMUSA, México D.F. 1987

- Piattini, Mario G. *et. al. Elementos y Herramientas en el Desarrollo de Sistemas de Información, una visión actual de la tecnología CASE.* Addison-Wesley Iberoamericana S.A., E.U.A. 1995.

- Pressman, Roger S. *Software Engineering, a practitioner approach*, tercera ed. McGraw Hill, U.S.A. 1992.
- Revista: Soluciones Avanzadas, marzo-abril de 1994.

- SARH, Subsecretaría Forestal. *Bases y principios del manejo integral forestal.* SARH México D.F. 1989.

- SARH, & Universidad de Helsinki, *Documento principal del sistema de conservación y desarrollo silvícola*, México D.F. 1992.

- Squire, Enid. *Introducción al diseño de sistemas,* Fondo Educativo Interamericano. México, 1984.

- Van Gigch, John P. *Teoría General de Sistemas*, Trillas, México, 1987.

- Winblad, L. Ann. *et. al. Software Orientado a Objetos*, Addison-Wesley Iberoamericana S.A. Wilmington, Delaware, E.U.A., 1993.

- Yourdon, Edward. *Análisis Estructurado Moderno*, Prentice Hall Hispanoamericana, México, 1993.

DIRECCIONES INTERNET:

- http://www.pcuniv.com/oboranal.htm
- http://www.tregistry.com/pobmeth.htm
- http://wwwis.cs.utwente.nl