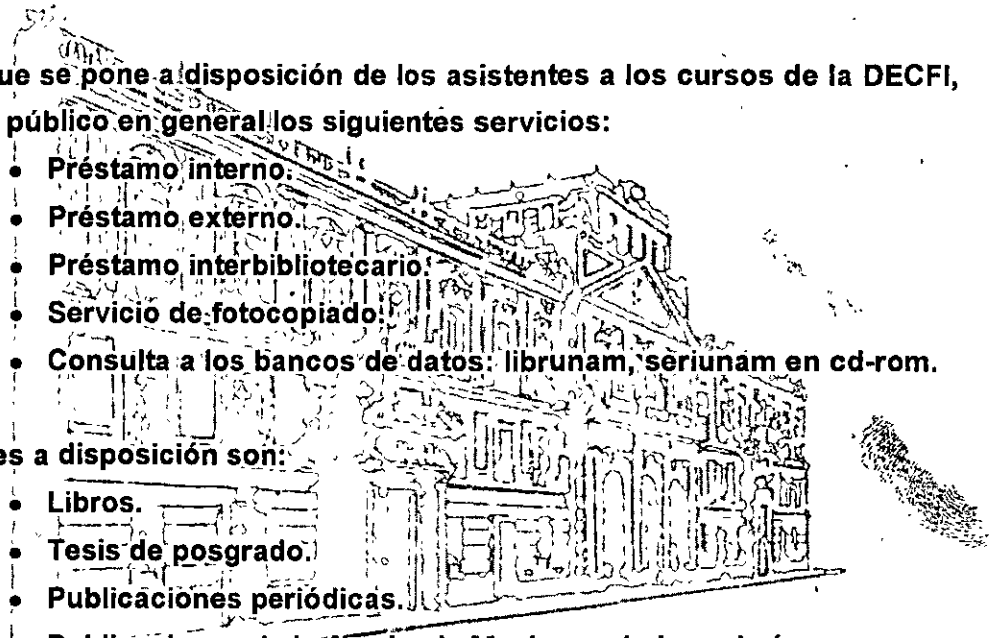
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" ING. BRUNO MASCANZONI "

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.



Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

- **Libros.**
- **Tesis de posgrado.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1988 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 14:30 y 16:00 a 17:30 de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

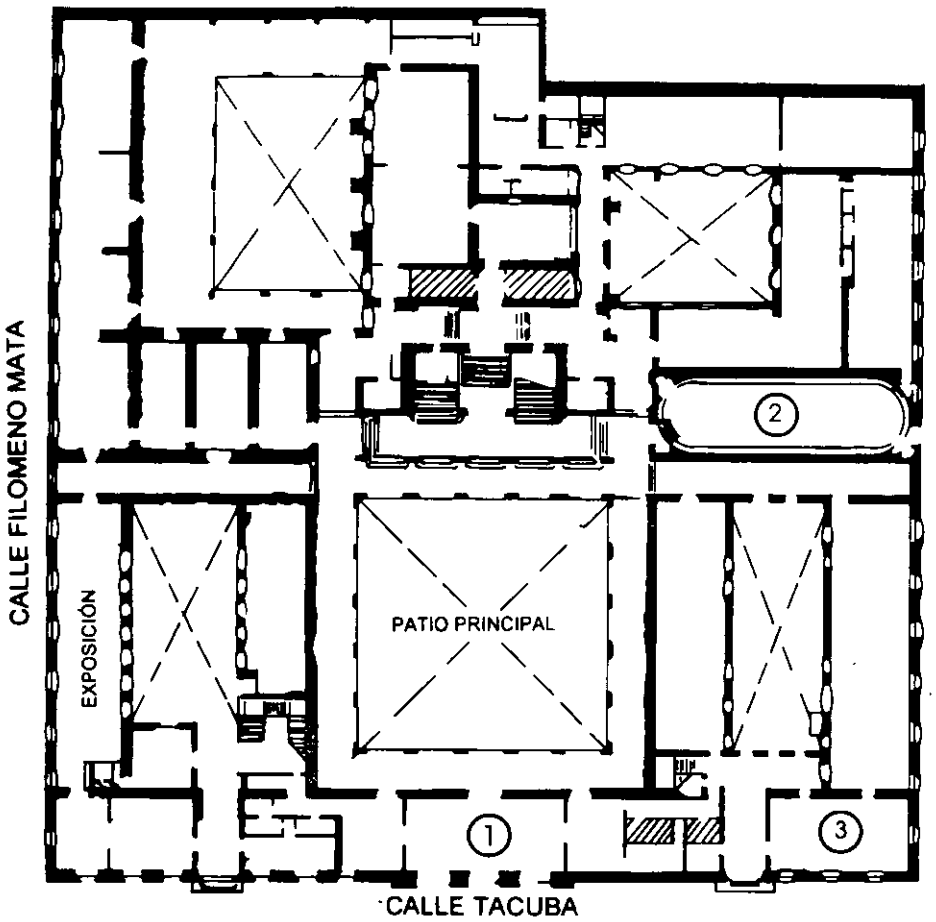
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

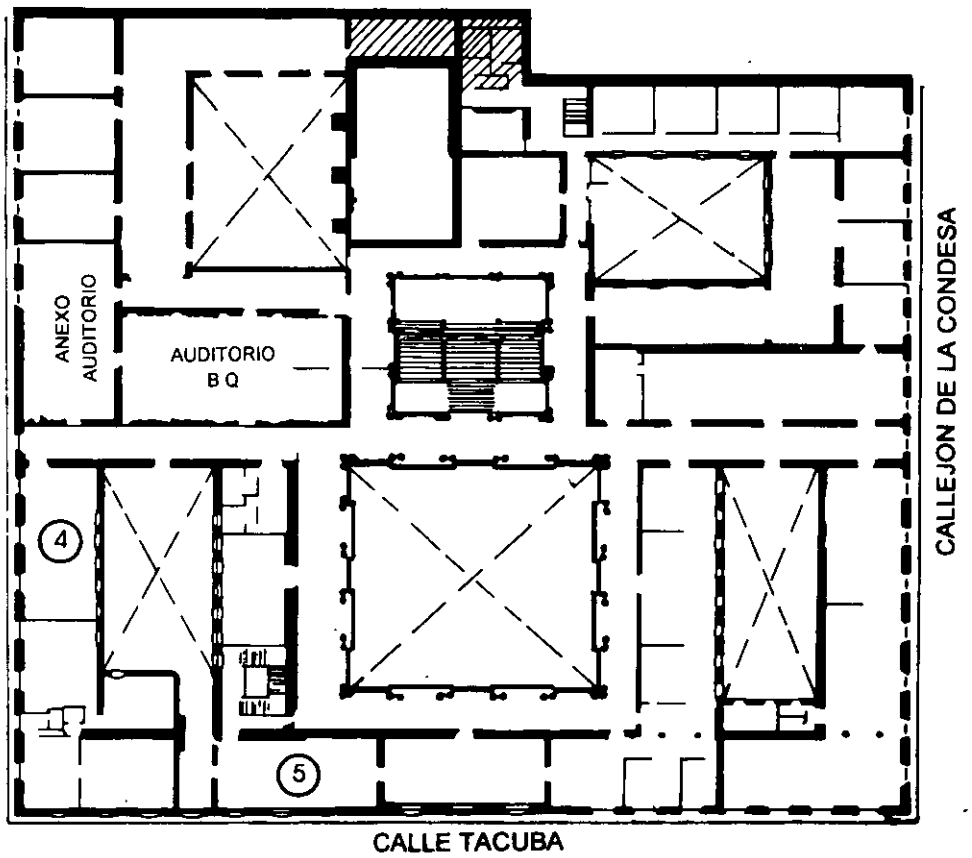
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

PALACIO DE MINERIA

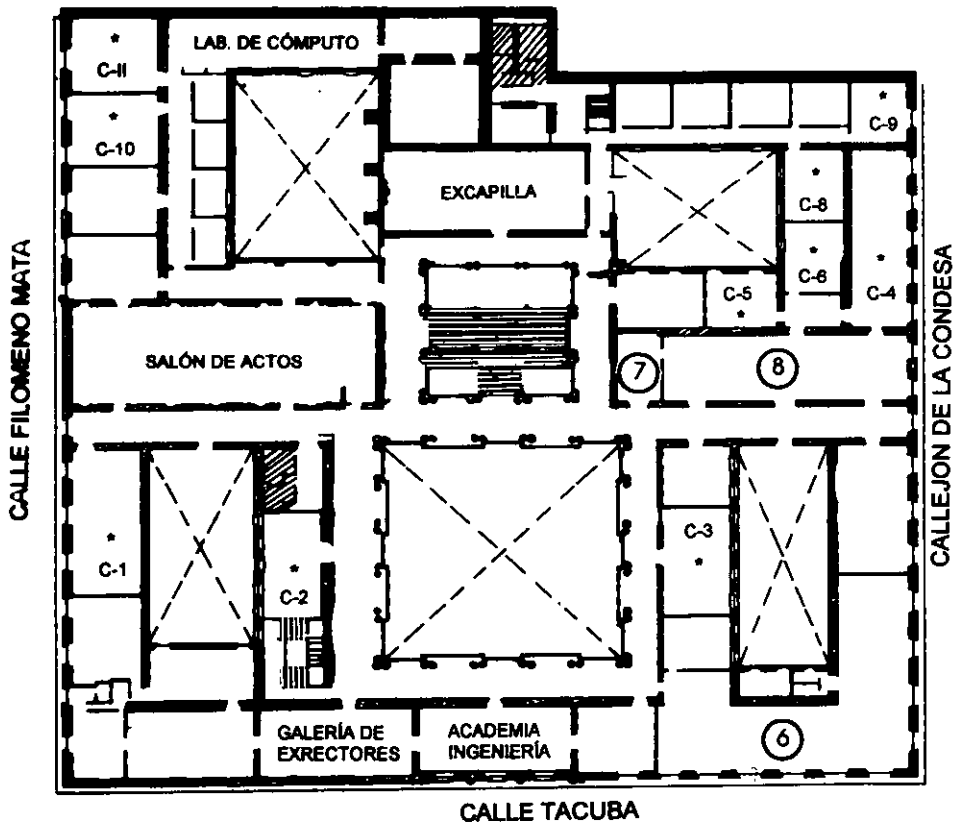


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



Ier. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACION CONTINUA
DEC

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: ***CC048 Diseño Orientado a Objetos para Programar en Java o C++***
FECHA: ***21 al 25 de agosto del 2000***

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
Yasmín Vega Hernández				
Jonathán Gamboa Beltrán				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso_____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

2. Medio a través del cual se enteró del curso:

Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

3. ¿Qué cambios sugeriría al curso para mejorarlo?

_____

_____

_____

_____

_____

_____

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ☐ NO ☐

5.¿Qué cursos sugiere que imparta la División de Educación Continua?

_____

_____

_____

_____

_____

_____

6. Otras sugerencias:

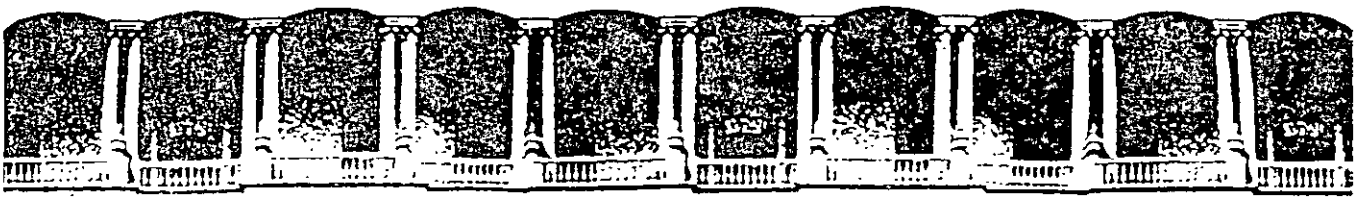
_____

_____

_____

_____

_____



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

MATERIAL DIDACTICO
DEL CURSO

DISEÑO ORIENTADO A OBJETOS PARA PROGRAMA EN JAVA O C ++

AGOSTO, 2000

I Introduccion.

I.1 Crisis del software.

Un constructor pensaría raramente en añadir un subsótano a un edificio ya construido de cíen plantas; hacer tal cosa sería muy costoso e indudablemente sería una invitación al fracaso. Asombrosamente, los usuarios de sistemas de software casi nunca se lo piensan dos veces a la hora de solicitar cambios equivalentes. De todas formas, argumentan, es simplemente cosa de programar.

Nuestro fracaso en dominar la complejidad del software lleva a proyectos retrasados, que exceden el presupuesto, y que son deficientes respecto a los requerimientos fijados. A menudo se llama a esta situación la crisis del software, pero, francamente, una enfermedad que ha existido tanto tiempo debe considerarse normal. Tristemente, esta crisis se traduce en el desperdicio de recursos humanos —un bien de lo más precioso—, así como en una considerable pérdida de oportunidad. Simplemente, no hay suficientes buenos desarrolladores para crear todo el nuevo software que necesitan los usuarios. Peor aún, un porcentaje considerable del personal de desarrollo en cualquier organización debe muchas veces estar dedicado al mantenimiento o la conservación de software geriátrico. Dada la contribución tanto directa como indirecta del software a la base económica de la mayoría de los países industrializados, y considerando en qué medida el software puede amplificar la potencia del individuo, es inaceptable permitir que esta situación se mantenga.

¿Cómo puede cambiarse esta imagen desoladora? Ya que el problema subyacente surge de la complejidad inherente al software, la sugerencia que se plantea aquí es estudiar en primer lugar cómo se organizan los sistemas complejos en otras disciplinas. Realmente, si se echa una mirada al mundo que nos rodea, se observarán sistemas con éxito dentro de una complejidad que habrá que tener en cuenta. Algunos de esos sistemas son obras humanas, como el transbordador espacial, el túnel bajo el Canal de la Mancha, y grandes organizaciones como Microsoft o General Electric. De hecho aparecen en la naturaleza sistemas mucho

más complejos aún, como el sistema circulatorio humano o la estructura de una planta.

I.2 Historia de la programacion orientada a objetos.

La orientación a objetos ha sido concepto establecido de software desde los 90, beneficiándose los programadores y los usuarios finales de la realización de este nuevo paradigma. Aunque el cambio del diseño y programación sobre procesos al diseño y programación sobre objetos puede parecer extremo, la programación orientada a objetos es una forma más natural de diseñar y modelar el software. Con la realización completa de la orientación a objetos, los usuarios finales pueden ser capaces de ampliar sus capacidades para modificar y quizá programar sus propias aplicaciones. Los programadores pueden diseñar aplicaciones más complejas en partes que son modulares e intercambiables.

Ciertamente la industria de software para computadores es propensa a la hipérbole. Aquellos creadores de software que han tenido años de experiencia con el paradigma de la orientación a objetos dirían qué parte de este dilema es debido a la resistencia a cambiar de los programadores que están familiarizados y se sienten cómodos con sus tradicionales lenguajes, herramientas, bases de datos y paradigmas de desarrollo en general. Todavía podríamos argumentar que dicha resistencia es conocida y está siendo vencida gradualmente a medida que las bases de datos, lenguajes y herramientas existentes que incorporan los mecanismos orientados a objetos permiten al creador de software cubrir lo nuevo y lo antiguo y experimentar los beneficios que la orientación a objetos aporta a la creación de software.

La orientación a objetos es un paradigma importante para los desafíos en el desarrollo del software lo largo de una amplia gama de tecnologías y saturará la próxima generación de arquitecturas de software. El paradigma mejora el proceso de desarrollo del software y hace aparecer nuevas y mejores aplicaciones.Esta capacidad se ha demostrado y asumido por los creadores de software más vanguardistas que disponen de arquitecturas objeto, base sobre la cual residirán las

futuras aplicaciones, hoy en construcción. De este modo, la orientación a objetos está incrustandose en las aplicaciones y herramientas de desarrollo de software que se utilizan actualmente, los entornos de desarrollo orientados a objetos están en manos de los principales creadores y ya proporcionan beneficios demostrables en términos de plazos de desarrollo, recursos necesarios de programación y posibilidad de construir nuevas generaciones del software existente.

I.2.1¿Por qué aparece ahora la orientación a objetos?

La orientación a objetos no es un concepto de nuevo cuño. De hecho, tiene por lo menos 25 años de antigüedad. Sus raíces pueden encontrarse en Noruega a finales de los años 60 en conexión con un lenguaje llamado Simula67, desarrollado por Kristen Nygaard y Ole-Johan Dahl en el Centro de Cálculo Noruego (Millikin 1989). Simula67 introdujo por primera vez los conceptos de clases, corrutinas y subclases, muy parecidos a los lenguajes orientados a objetos de hoy en día.

Posteriormente, a mitad de la década de los 70, los científicos del Centro de Investigación Palo Alto de Xerox (Xerox PARC) crearon el lenguaje Smalltalk, el primer lenguaje orientado a objetos consistente y completo. En Smalltalk, cada elemento del lenguaje fue realizado como un objeto (Goldberg y Robson 1983). Con Smalltalk, cada aspecto del lenguaje, el entorno de programación y la cultura que lo rodeaba eran orientados a objetos. Incluso hoy, Smalltalk se considera el más puro de los lenguajes orientado a objetos. Los desarrollos Simula y Smalltalk impulsaron gran parte del trabajo que actualmente sucede. Simula67 demostró el poder de modelación de un lenguaje de programación basado en clases, así como la idea de que los datos y operaciones debían almacenarse juntos. Quedó reconocido que se podía ahorrar esfuerzo de programación si las propiedades comunes de los objetos podían ser preprogramadas.

Hace dos decadas, la orientación a objetos era lenta para penetrar la corriente principal de la comunidad de las computadoras. Esta lenta migración era debida a un cierto número de factores. Aunque bien conocidos en los círculos universitarios, Simula67 y Smalltalk fueron relativamente inaccesibles a la corriente principal de la

comunidad de computadoras hasta los años 80. Por ejemplo, el trabajo inicial en Smalltalk no fue dado a conocer públicamente fuera de Xerox PARC hasta el ejemplar de Agosto de 1981 de la revista Byte. Las necesidades de plataformas especializadas de cálculo hicieron también poco atractivo económicamente el empleo de estos lenguajes iniciales a los creadores de software empresarial y comercial. Muchos desarrolladores, al conocer por primera vez Smalltalk, lo consideraron más como un sistema de ventanas que como un paradigma revolucionario de programación. El fracaso de apreciar el paradigma de la programación orientada a objetos en Smalltalk está demostrado por el desarrollo posterior de muchos interfaces que utilizan ventanas e iconos sin permitir a los usuarios modificar estos objetos (Thomas 1989).

En los años 80, C se convirtió en un lenguaje de desarrollo muy popular, no sólo en las microcomputadoras sino en la mayoría de las arquitecturas y entornos de computación. Al principio de la década de los 80, Bjarne Stroustrup de los Laboratorios Bell de AT&T amplió el lenguaje C para crear C++, un lenguaje que soporta la programación orientada a objetos (Stroustrup 1986). Posteriores mejoras en herramientas y lanzamientos comerciales del lenguaje C++ por AT&T, y otros fabricantes, justifican buena parte de la atención general hacia la programación orientada a objetos en la comunidad de software. Con C++, los programadores eran capaces de aprender el paradigma de la orientación a objetos en un léxico popular y conocido, sin tener que invertir en nuevos y diferentes entornos y lenguaje de computación.

Los obstáculos precio/rendimiento que evitaban el empleo general de la tecnología orientada a objetos también han caído en el camino. Las configuraciones de los potentes computadores personales de hoy en día han satisfecho las exigencias básicas de estaciones de trabajo de rendimiento mayor, presentaciones gráficas de calidad más alta, y soportan de forma completa entornos de desarrollo con multitud de herramientas; y la introducción de los entornos integrados orientados a objetos, específicamente el Macintosh en 1984 y la máquina NeXT en 1988, fueron hitos significativos en la introducción de la orientación a objetos en la corriente principal de la computación.

Sin embargo, la creciente complejidad tecnológica es el acelerador oculto para el uso del paradigma orientado a objetos. La orientación a objetos proporciona una forma mejor de gestionar la complejidad tecnológica. Los programadores que utilizan las herramientas actuales de desarrollo para crear aplicaciones indican que sin estratos o capas de abstracción, el desarrollo de aplicaciones se estrangula. La orientación a objetos permite la programación en niveles más altos de abstracción - desde el objeto a la clase, hasta la biblioteca de clases y finalmente a los completos marcos estructurales de aplicación.

Por último, la industria del software está deseosa de intentar un método mejor para el desarrollo de software. La industria del software ha llegado a estar atascada (algunos la han descrito como "tumbada en la playa") debido a la falta de procesos mejores. La orientación a objetos promete proporcionar a los desarrolladores de software comercial la capacidad para el desarrollo acelerado de programas, un codigo ampliable y productos que pueden comercializarse en una base mayor de clientes.

I.3 Ventajas de los lenguajes orientados a objetos.

Un lenguaje de programación que soporta el paradigma orientado a objetos beneficia al desarrollador de software proporcionando una forma natural de modelar el fenómeno del complejo mundo-real. Aunque el paradigma orientado a objetos de los objetos, métodos y mensajes es con frecuencia difícil de entender para los programadores acostumbrados a la metáfora tradicional de procedimientos y datos, la orientación a objetos ofrece realmente un modelo más natural del mundo real. Programar ya no significa solamente escribir líneas de código sino desarrollar modelos utilizando clases. Con la programación orientada a objetos, los programas tienen menos líneas de código, menos sentencias de bifurcación y módulos que son más comprensibles porque reflejan una relación unívoca entre el modelo objeto y el conceptual.

Las bibliotecas de clases predefinidas, un componente de los lenguajes

orientados a objetos maduros, aumentan las ventajas de utilizar lenguajes orientados a objetos. Buena parte de la programación hecha con Smalltalk, por ejemplo, puede hacerse con objetos y mensajes que ya existen en la biblioteca Smalltalk. El trabajo de programación consiste en encontrar en la biblioteca Smalltalk los objetos y mensajes adecuados y combinarlos en el orden apropiado Este empleo de una biblioteca de clases es similar al uso de las librerías de funciones, como las que se utilizan en C, pero hay una diferencia significativa La biblioteca de funciones de C es relativamente fija. Con la librería Smalltalk y las bibliotecas que acompañan a otros lenguajes orientados a objetos, los nuevos miembros pueden crearse fácilmente por medio del empleo de la herencia.

El método general de reducir el código mediante la programación de las diferencias con herencia es una de las tácticas clave de la programación orientada a objetos y es una capacidad única de los lenguajes orientados a objetos.

II Conceptos Básicos.

II.I Mecanismos básicos.

Los mecanismos básicos de la orientación a objetos son los objetos, mensajes y los métodos, clases y variables instancia (o modelo) y herencia. Todos los sistemas que merecen la descripción de orientado a objetos contienen estos mecanismos esenciales, aunque los mecanismos pueden no estar realizados (o denominados) exactamente de la misma forma.

II.1.1 Objetos.

Se definirá un objeto como un concepto, abstracción o cosa con limites bien definidos y con significado a efectos del problema que se tenga entre manos. Los objetos tienen dos propósitos: promover la comprensión del mundo real y proporcionar una base practica para la implementación por computadora. La descomposición de un problema en objetos depende del juicio y de la naturaleza del problema. No existe una única representación correcta.

Un programa tradicional consta de procedimientos y datos. Un programa orientado a objetos consta solamente de objetos que contienen tanto los procedimientos como los datos. Por decirlo de otra forma, los objetos son módulos que contienen los datos y las instrucciones que operan sobre esos datos. Así, dentro de los objetos residen los datos de los lenguajes convencionales, como por ejemplo, números, matrices (arrays o arreglos), cadenas de caracteres y registros, así como cualquier función, instrucción o subrutina que opere sobre ellos. Los objetos, por tanto, son entidades que tienen atributos (datos) y formas de comportamiento (procedimientos) particulares.

Los objetos llevan los nombres de los elementos de interés desde el dominio de la aplicación. Por ejemplo, en una aplicación de procesamiento de textos, es probable que un objeto sea llamado párrafo. En una aplicación de contabilidad, la hoja balance de junio es un objeto probable. Desde la perspectiva del usuario, los objetos proporcionan el comportamiento deseado. Un párrafo puede aceptar revisión y realinear sus márgenes. La hoja balance del mes de junio puede imprimirse o consolidarse con las de otros meses para constituir un informe cuatrimestral. Desde la perspectiva del programador, los objetos son módulos de una aplicación que funcionan juntos para proporcionar una funcionalidad general.

Es importante señalar que todos los objetos poseen su propia identidad y se pueden distinguir entre si. Dos manzanas del mismo color, forma y textura siguen siendo manzanas individuales. El termino identidad significa que los objetos se distinguen por su existencia inherente y no por las propiedades descriptivas que puedan tener.

Las aplicaciones pueden constar de diferentes clases de objetos. Un objeto activo es aquel que se comprende su propio hilo de control, mientras que un objeto pasivo no. Los objetos activos suelen ser autónomos lo que quiere decir que pueden exhibir algún comportamiento sin que ningún otro objeto opere sobre ellos. Los objetos pasivos, por otra parte, solo pueden padecer un cambio de estado cuando se actúa explícitamente sobre ellos. De este modo los objetos activos de un sistema sirven como raíces de control.

II.1.2 Mensajes y Métodos.

En la mayoría de los lenguajes de programación orientados a objetos las operaciones que los clientes pueden realizar sobre un objeto suelen declararse como métodos, que forman parte de la declaración de la clase. El paso de mensajes es una de las partes de la ecuación que define el comportamiento de un objeto; la definición de comportamiento también recoge que el estado de un objeto afecta a si mismo a su comportamiento. A diferencia de los elementos de datos pasivos en los sistemas tradicionales, los objetos tienen la posibilidad de actuar. La acción sucede cuando un objeto recibe un mensaje, que es, una solicitud que pide al objeto que se comporte de alguna forma. Cuando se ejecutan los programas orientados a objetos, los objetos reciben, interpretan y responden a mensajes procedentes de los objetos. Por ejemplo, cuando un usuario solicita que un objeto llamado documento se imprima a sí mismo, el documento puede enviar un mensaje al objeto impresora solicitando un lugar en la cola de impresión; el objeto impresora puede devolver un mensaje al documento solicitando información de formato, y así sucesivamente Los mensajes pueden contener información para clarificar una solicitud; por ejemplo, el mensaje solicitando que un objeto se imprima a sí mismo podría incluir el nombre de la impresora. Finalmente, el emisor del mensaje no necesita conocer la forma en que el objeto receptor está llevando a cabo la solicitud. En otras palabras, cuando el objeto documento recibe el mensaje imprimir, documento sabe exactamente lo que tiene qué hacer. El objeto que envía el mensaje ni sabe ni le importa cómo se realiza la impresión, solamente conoce que está sucediendo.

El conjunto de mensajes al que un objeto puede responder se llama protocolo del objeto. El protocolo para un icono puede constar de mensajes invocados por la pulsación del botón del ratón cuando el usuario localiza un puntero sobre un icono.

Los métodos pueden enviar también mensajes a otros objetos solicitando acción o información.

Al igual que las "cajas negras" de la ingeniería, la estructura interior de un objeto

está oculta a usuarios y programadores. Los mensajes que recibe el objeto son los únicos contactos que conectan al objeto con el mundo exterior. Solamente un método del propio objeto puede disponer de los datos del interior del mismo para su manipulación. Estas características de los objetos confieren a la orientación a objetos su ventaja: La orientación a objetos fomenta la modularidad haciendo muy claras las fronteras entre objetos, explícita la comunicación entre los mismos y ocultos los detalles de la realización.

Cuando se ejecuta un programa orientado a objetos, ocurren tres sucesos. En primer lugar, se crean los objetos cuando se necesitan. Segundo, los mensajes se mueven de un objeto a otro (o desde el usuario a un objeto) a medida que el programa procesa internamente información o responde a la entrada del usuario. Finalmente, se borran los objetos cuando ya no son necesarios y se recupera memoria.

II.1.3 Clases, subclases y objetos.

Muchos objetos diferentes pueden actuar de formas muy similares. Una clase es una abstracción que describe propiedades importantes para una aplicación y que ignora el resto. Una clase consta de métodos y datos que resumen las características comunes de un conjunto de objetos. La posibilidad de abstraer métodos y descripciones de datos comunes de un conjunto de objetos y almacenarlos en una clase es esencial para la potencia de la orientación a objetos. Definir clases significa situar código reutilizable en un depósito común en lugar de volver a expresarlo una vez y otra. En otras palabras, las clases contienen los anteproyectos para crear objetos. Finalmente, la definición de una clase ayuda a clarificar la definición de un objeto: un objeto es un modelo o instancia de una clase.

Los objetos se crean cuando se recibe un mensaje solicitando creación por la clase padre. El nuevo objeto toma sus métodos y datos de su clase padre. Los datos son de dos formas, variables de clase y variables modelo o de instancia. Las variables de clase tienen valores almacenados en una clase; las variables instancia tienen valores asociados únicamente con cada instancia u objeto creado a partir de una clase.

A título de ejemplo, considere cómo un programador podría designar una aplicación de procesamiento de textos en forma orientada a objetos. En primer lugar el programador identifica las entidades de interés. Los párrafos, por ejemplo, son objetos potenciales. Justificar es un método común para todos los párrafos. Tipodeletra (Fuente) es una variable de clase con el valor helvética. Finalmente, texto es una variable modelo con valores únicos para cada objeto. Es útil crear una clase llamada párrafo para guardar esta información común. La clase párrafo proporciona entonces un anteproyecto para la construcción de objetos. Aunque los datos específicos de cada objeto (ej.: las palabras del párrafo, el tipo de letra o fuente, el interlineado o espaciado) pueden variar, todos los objetos de la clase párrafo comparten métodos y variables de clase comunes.

Una clase puede también resumir elementos comunes para un conjunto de subclases. En el caso del ejemplo del procesamiento de textos, el programador puede considerar después tabla para que sea otra clase además de párrafo. Pensando un poco más, el programador se da cuenta que párrafo y tabla comparten algunas propiedades con una clase más abstracta, texto. De forma similar, texto y gráfico pueden convertirse en subclases de una clase con carácter aún más general, documento ilustra esta jerarquía de las clases del procesamiento del documento. Utilizando subclases, los programadores orientados a objetos describen las aplicaciones como conjuntos de módulos generales o abstractos. Los métodos y datos comunes se elevan tan alto como sea posible de forma que sean accesibles a todas las subclases relacionadas.

A veces se denomina a las subclases como clases derivadas. En otras ocasiones los términos padre e hija se utilizan para indicar la relación entre una clase y una subclase. Las clases padre están localizadas por encima de las clases hijas en la jerarquía. Las clases más altas en la jerarquía se denominan las superclases.

Hasta ahora, esta descripción de un diseño orientado a objetos ha sido "ascendente". Es decir, la descripción pone el énfasis en la forma en que los componentes (objetos) son abstraídos para llegar a ser clases y superclases. De

hecho. el término "descendente" ("top-down") describe con mayor precisión el método seguido por muchos programadores de la orientación a objetos. Normalmente comienzan su trabajo con una biblioteca de clases que contiene generalmente módulos útiles de programación. Utilizando clases predefinidas como punto inicial, los programadores escriben nuevas subclases que adaptan las clases de empleo general de la biblioteca a los requisitos funcionales particulares de la aplicación.

Una biblioteca de clases específicas para una aplicación se denomina un marco estructural "framework". Los marcos estructurales difieren de las bibliotecas de clases en su distinto grado: un marco estructural es una biblioteca de clases ajustada especialmente para una determinada categoría de aplicaciones, por ejemplo, un marco estructural constructor de interfaces. Construir y adaptar aplicaciones a partir de marcos estructurales es más rápido y fácil que empezar con bibliotecas de clases genéricas. Asimismo, un marco estructural no será normalmente útil fuera del campo de la aplicación ya que contiene clases específicas para la aplicación.

II.1.4 Herencia.

La herencia es el mecanismo para compartir automáticamente métodos y datos entre clases, subclases y objetos. En términos generales se puede definir una clase que después se ira refinando sucesivamente para producir subclases. Todas las subclases poseen, o heredan, todas y cada una de las propiedades de su superclase y añaden además, sus propiedades exclusivas. No es necesario repetir las propiedades de las superclases en cada subclase. La herencia permite a los programadores crear nuevas clases programando solamente las diferencias con la clase padre. Cuando un programador declara que párrafo es una subclase de texto, por ejemplo, todos los métodos y variables modelo asociados con texto son heredados automáticamente por párrafo. Si la clase texto contiene métodos que son inadecuados para la subclase párrafo, entonces el programador puede obviar estos métodos escribiendo unos nuevos y almacenándolos como parte de la clase párrafo.

Debido a la herencia, los programas orientados a objetos constan de taxonomías, árboles o jerarquías de clases que, por medio de la sub clasificación, llegan a ser más específicas. Las clases proporcionan los anteproyectos para las subclases o para los objetos relacionados con una aplicación.

Herencia simple y múltiple son dos tipos de mecanismos de herencia utilizados normalmente en la programación orientada a objetos. Con la herencia simple, una subclase puede heredar datos y métodos de una clase simple así como añadir o sustraer comportamiento por sí misma. La herencia múltiple se refiere a la posibilidad de una subclase de adquirir los datos y métodos de más de una clase. La herencia múltiple es útil al construir comportamiento compuesto a partir de más de una rama de una jerarquía de clases.

En resumen, los mecanismos básicos de la orientación a objetos conducen a una particular visión sobre el concepto de dar forma al mundo. Los elementos y su comportamiento se identifican como objetos. El comportamiento se realiza con métodos y datos almacenados en el objeto. Los mensajes obtienen el comportamiento de un objeto invocando un método del mismo.

Los objetos con métodos y variables modelo comunes se reúnen en una clase. Las clases se organizan en jerarquías y los mecanismos de herencia proporcionan automáticamente a cada subclase los métodos y datos de las clases padre. Las subclases se crean programando las diferencias entre las clases disponibles en una librería y los requisitos particulares de la aplicación.

II.2 Conceptos clave.

Los mecanismos básicos señalados anteriormente forman la base del paradigma de la orientación a objetos. Cuatro conceptos clave que resumen las ventajas del método orientado a objetos son la encapsulación, la abstracción, el polimorfismo y la persistencia.

II.2.1 Encapsulación.

La encapsulación (denominada también ocultamiento de información) consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás. La encapsulación evita que el programa llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin afectar a las aplicaciones que la utilizan. Quizá sea necesario modificar la implementación de un objeto para mejorar el rendimiento, corregir un error, consolidar el código o para hacer un transporte a otra plataforma.

La encapsulación no es exclusiva de la programación orientada a objetos pero la capacidad de combinar la estructura de datos y el comportamiento en una única entidad hace que la encapsulación sea aquí mas limpia y potente que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

II.2.2 Abstracción.

La abstracción consiste en centrarse en los aspectos esenciales inherentes de una entidad, e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir como debería ser implementado. El uso de la abstracción mantiene nuestra libertad de tomar decisiones durante el mayor tiempo posible evitado comprometernos de forma prematura con ciertos detalles. La mayoría de los lenguajes modernos proporcionan abstracción de datos pero la capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional. El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño o de implementación antes de haber comprendido el problema. Un uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de

datos y documentación. Un estilo de diseño independiente del lenguaje pospone los detalles de programación hasta la fase final, relativamente mecánica del desarrollo.

II.2.3 Polimorfismo.

Los objetos actúan en respuesta a los mensajes que reciben. El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo. Con el polimorfismo un usuario puede enviar un mensaje genérico y dejar los detalles exactos de la realización para el objeto receptor. El mensaje imprimir, por ejemplo, al ser enviado a una figura o diagrama invocará diferentes métodos de impresión que en el caso de enviar el mismo mensaje imprimir a un documento de texto.

El polimorfismo está fomentado por la maquinaria de la herencia. Es muy normal almacenar los protocolos para funciones de utilidad como "imprimir" en la posición más alta que sea posible dentro de la jerarquía de clases. Las variaciones necesarias en el comportamiento "imprimir" se almacenan en niveles más bajos de la jerarquía para sobrescribir los métodos más generales cuando sea necesario. De esta forma, los objetos están listos y pueden responder apropiadamente a mensajes de utilidad como "imprimir" mientras que el método que realiza la función de impresión puede existir en la clase inmediata del objeto o varios niveles por encima de la clase del objeto.

II.2.4 Persistencia.

La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo durante el cual se asigna espacio y permanece accesible en la memoria de la computadora. En la mayoría de los lenguajes orientados a objetos, se crean modelos de clases a medida que el programa se ejecuta. Algunos de estos modelos se necesitan solamente por un breve período de tiempo. Cuando un objeto ya no es necesario, es destruido y recuperado el espacio de memoria que tenía asignado. La recuperación automática del espacio de memoria se denomina normalmente recolección de basura.

Después de haber ejecutado un programa orientado a objetos, los objetos ensamblados normalmente no se almacenan; es decir, los objetos dejan de ser persistentes. Una base de datos orientada a objetos mantiene una distinción entre objetos creados solamente para el tiempo de duración de la ejecución y aquellos pensados para almacenamiento permanente. Los objetos almacenados permanentemente se denominan persistentes.

III Introducción al Analisis.

III.1 Historia del análisis y diseño orientados a objetos.

Habida cuenta de la importancia de comprender y comunicar los requisitos de un sistema antes de programarlo, los diseñadores de sistemas han continuado buscando métodos mejorados para esta temprana fase del proceso de desarrollo. Los métodos antiguos descansaban en gran medida en descripciones textuales realizadas sobre el papel. Estos métodos conducían no sólo a la ambigüedad sino que también dificultaban las modificaciones y eran incapaces de apoyar el diseño de sistemas muy grandes. El análisis estructurado, basado principalmente en descomponer componentes funcionales de un sistema fue desarrollado en los años 60 e introdujo un método definitivo y más manejable para el análisis de sistemas. El análisis de sistemas orientado a objetos es un nuevo método que realza la definición de las características y el comportamiento dentro de un sistema de objetos.

III.2 Análisis orientado a objetos.

El modelado de información, popularizado por Peter Chen en la década de los ochenta, es el precursor más próximo del análisis y diseño orientados a objetos. El resultado final de análisis de modelado de información es un diagrama de entidad-relación, desarrollado a base de listar atributos, clasificarlos en categorías de entidades y, finalmente, agregando las relaciones existentes entre ellos. El modelo inicial se perfecciona posteriormente con una jerarquía de sub-tipos y objetos asociativos. Aunque este método está estrechamente relacionado con el análisis orientado a objetos, no existe encapsulación de datos; es decir, los requisitos de

procesamiento de cada objeto y de sus atributos no se tratan como una entidad combinada. El método de modelado de información tampoco permite la herencia ni el paso de mensajes.

El primer material publicado sobre el análisis orientado a objetos provino de Sally Shlaer y Stephen Mellor (1988). comienza por definir objetos y atributos A continuación se definen los ciclos de vida de los objetos en modelos de estado para capturar los sucesos que actúan sobre los objetos. El último paso es la definición de procesos, basada en los objetos y sus ciclos de vida. A diferencia de los métodos de descomposición funcional y de suceso-respuesta, el método orientado a objetos se traduce en un mínimo código derivado de los datos, que permanece estable incluso ante el cambio de requisitos.

Los tres componentes de un sistema -proceso (lo que se está haciendo), datos (aquello sobre lo que el proceso se está ejecutando), y control (en qué momento se realiza)- reciben diferente prioridad y énfasis en los métodos de descomposición funcional, suceso-respuesta, y aquellos orientados a objetos.

En un sistema orientado a objetos no recae el énfasis en las transformaciones de entradas en salidas, sino en el contenido de entidades, en los objetos. El criterio para agrupar funciones no es el proceso; más bien se trata de agrupar métodos cuando estos funcionan sobre una misma abstracción de datos. Es fácil que métodos contiguos en una secuencia residan en objetos diferentes. Es el paso de mensajes entre objetos lo que determina la secuencia de funcionamiento.

III.3 Ventajas del análisis y diseño orientados a objetos.

Pocas de las herramientas actuales de análisis y diseño estructurados contemplan los mecanismos de la programación orientada a objetos. Algunas metodologías incluidas en herramientas automatizadas se promocionan como orientadas a objetos. En la mayoría de los casos los objetos en cuestión son procesos, módulos, o solo grandes objetos binarios (blobs), normalmente

diagramas. Para apoyar la programación orientada a objetos, el análisis y el diseño se deben centrar en:

La identificación de objetos y la definición de clases, la organización jerarquizada de clases,la reutilización de clases, y la construcción de marcos estructurales de aplicación a partir de librerías de clases.

No solo no se aplican los métodos estructurados al diseño de software orientado a objetos, sino que, en realidad, no existe un método de diseño orientado a objetos que sea ampliamente aceptado. Quizá una de las razones es que el propio paradigma de la programación orientada a objetos contiene importantes técnicas de diseño. El énfasis en la reutilización, modularidad y encapsulación, así como la fuerte ligazón existente entre diseño y código, forman parte integral de los sistemas orientados a objetos. Como los sistemas orientados a objetos contienen una buena cantidad de la información de diseño, los procesos de realización y diseño se entremezclan más estrechamente. La programación orientada a objetos permiten que el diseño de un programa brille a través del código. Se puede expresar un buen diseño inicial como una versión anterior del código, o el código como una versión anterior del diseño.

Independientemente de la estrecha unión entre diseño y programación, sigue teniendo importancia la bondad del análisis y el diseño que se traducirá en mejores sistemas. Las aplicaciones orientadas a objetos con un pobre diseño tienen todas las posibilidades de contar con una complejidad inherente más allá de la capacidad del programador individual. Un diseño orientado a objetos comienza con un conjunto de definiciones de clases. Cada clase dispone de un conjunto de métodos que define y una lista de objetos a los que sus modelos pasan mensajes. Un diseño queda completo cuando se define cada clase, método y mensaje. Este proceso de diseño suele ser incremental y los diseñadores rara vez comienzan partiendo de la nada; en lugar de ello, suelen partir de, y reutilizar, bibliotecas de clases preexistentes, adaptando clases específicas a las necesidades del sistema. Por tanto, una metodología para esta aproximación debe proporcionar reglas para identificar, definir y organizar las clases y los métodos y mensajes que las

acompañan. También ha de proporcionar estrategias para organizar librerías de clases y directrices para la construcción de aplicaciones para marcos estructurales o bibliotecas de clases existentes.

El análisis y el diseño orientados a objetos ofrece importantes ventajas. Por una parte un buen diseño puede asegurar la realización de las ventajas máximas de un lenguaje de programación orientado a objetos, sobre todo cuando no se cuenta con buenas bibliotecas de clases preexistentes. En efecto, incluso cuando no se realiza el diseño mediante un lenguaje orientado a objetos, un diseño orientado a objetos puede aportar ciertas ventajas de los sistemas orientados a objetos a lenguajes que no permiten la aplicación completa del concepto. Un diseño orientado a objetos también suele producir menos código y más reutilizable que el intensivo proceso del método de descomposición procedimental.

Además, el método de diseño orientado a objetos tiende a producir sistemas flexibles a los cambios. Fija su atención en los elementos del sistema que son más estables: los objetos. Esta estabilidad y flexibilidad se encuentra presente en el propio diseño. Cuando hay que introducir un cambio, su exclusiva propiedad de herencia permite la reutilización y la ampliación del modelo existente.

La modularidad del método orientado a objetos beneficia también al equipo de desarrollo. Como los datos y procesos están localizados en los objetos, es fácil tener varios equipos trabajando independientemente en diferentes partes del diseño. Por último, el método de diseño orientado a objetos es más intuitivo, no solo para el analista sino también para el usuario final, ya que su enfoque de organización y comprensión del problema resulta natural; está basado en la forma de pensar de los seres humanos.

III.4 Proceso del análisis y diseño orientados a objetos.

El resultado de un diseño orientado a objetos es una jerarquía de clases. Cada clase es un módulo separado con sus propias estructuras de control y datos. Se puede ver la extensión del problema de forma más natural y realista como un

conjunto de objetos y métodos asociados. Los elementos iniciales de un diseño orientado a objetos son los propios objetos. Posteriormente, a medida que se identifican aspectos comunes, los objetos se van agrupando en clases que a su vez serán subclases de clases más abstractas. El nivel superior de abstracción se conoce normalmente como marco estructural. Un marco estructural es un conjunto de clases que expresa un diseño para una familia de aplicaciones relacionadas entre sí. Los métodos estructurados y sus gráficos de estructuras definen una aplicación como un juego secuencial de módulos interdependientes que comparten datos. Los métodos de objetos definen un conjunto de módulos de comunicación independientes con visibilidad limitada entre ellos.

En esencia, el diseño orientado a objetos consta de cuatro pasos fundamentales:

1. Identificación y definición de objetos y clases.
2. Organización de relaciones entre las clases.
3. Extracción de estructuras en una jerarquía de clases.
4. Construcción de bibliotecas de clases y marcos estructurales de aplicación reutilizables.

Al igual que el método de prototipos aplicado a los sistemas tradicionales, el diseño orientado a objetos es cíclico. Los programadores comienzan con un conjunto de clases, las amplían, las modifican y las encajan formando un prototipo de una aplicación. La interacción con los usuarios finales hace que el prototipo sea revisado y, a continuación, los analistas y programadores se vuelven a hacer cargo del trabajo. A lo largo de todo este ciclo se vuelven a definir y organizar las clases, y cada vez que se observa que una clase es lo suficientemente genérica como para ser reutilizada en aplicaciones futuras, se añade a las bibliotecas de clases estándar. Posteriormente, los diseñadores de marcos estructurales identifican similitudes entre aplicaciones y desarrollan marcos estructurales de clases que son útiles como soluciones de programación para problemas futuros similares.

III. 5 Encontrando clases y Objetos..

III.5.1 Identificación y definición de objetos.

El diseño de un sistema orientado a objetos comienza con los objetos. Encontrarlos es quizá el principal desafío del diseño y el análisis orientados a objetos. Se han utilizado varios métodos para identificar objetos, incluyendo la inspección gramatical de documentos y la derivación a partir de diagramas de flujo de datos y de relación de entidades.

Grady Booch (1983) dio origen al método gramatical y sugiere que el diseñador comience con una descripción en prosa del sistema deseado y vea los nombres como identificadores potenciales de las clases de objetos. Los verbos, por otro lado, identifican a los métodos. La lista resultante de clases (nombres) y métodos (verbos) se utilizará para comenzar el proceso de diseño.

A continuación se muestra un ejemplo del método léxico de Booch para el diseño de sistemas orientado a objetos. Su metodología, aunque ligada al lenguaje Ada, es útil para los programadores orientados procedimentalmente que quieran conseguir una perspectiva orientada a objetos. La metodología de Booch comienza por una definición del problema y una descripción de la solución, como se muestra en el ejemplo siguiente:

- Definición del problema: Desarrolla un programa muy sencillo de procesamiento de texto.

- Descripción de la solución: El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se archivan en un directorio del usuario. Los usuarios pueden imprimir o visualizar documentos. Se pueden cambiar los documentos. También se pueden borrar del directorio del usuario.

El paso siguiente consiste en identificar los posibles objetos subrayando los sustantivos (y las frases sustantivadas), tal como se muestra a continuacion.

El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se archivan en un directorio de usuario. Los usuarios pueden imprimir o visualizar documentos. Se pueden cambiar los documentos. También se puede borrar del directorio de usuario.

Documento y directorio parecen ser conceptos clave y, por tanto, objetos. Se debe observar que no todos los nombres que aparecen en una descripción inicial del sistema terminarán por convenirse en objetos. Aunque el método gramatical es una buena forma de desarrollar una lista inicial de posibles clases, generará un cierto número de conceptos que no pertenecen al sistema a modelar y no tienen que incorporarse al software. Nuestra descripcion contiene un ejemplo típico de esto en el nombre usuario.

Una vez identificados los objetos, Booch sugiere la asociación de atributos con cada uno de ellos. En el ejemplo del procesamiento de texto, los objetos y sus atributos son los siguientes:

Objeto	Atributos		
Documento	se	puede	crear
	se	puede	borrar
	se	puede	grabar (en disco)
	se	puede	imprimir
	se	puede	visualizar
	se	puede	cambiar
Directorio	Contiene uno o más documentos los documentos se pueden grabar en el directorio y se pueden borrar del mismo		

El paso siguiente del proceso de Booch consiste en identificar los métodos posibles subrayando los verbos, de la forma siguiente:

El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se graban o archivan en un directorio de usuario. Los usuarios pueden imprimir o visualizar documentos. Pueden cambiar los documentos. También se pueden borrar del directorio de usuario.

A continuación se recogen estos métodos con sus objetos correspondientes, de la forma siguiente:

Objeto	Método	
Documento	crear	guardar
	imprimir	visualizar
	cambiar	borrar
Directorio	archivar	borrar

La definición de las interfaces entre objetos es el paso final del proceso de Booch. Una vez más se crea una descripción escrita. Para el ejemplo anterior, la descripción podría ser la siguiente:

Se realiza el sistema como dos clases, documento y directorio.

La clase documento contiene una variable instancia llamada documentid (su propio nombre) y los métodos siguientes: crear, archivar, visualizar, cambiar y borrar. La clase directorio contiene directorid y los métodos archivar y borrar.

Booch proporciona una estrategia sencilla para identificar objetos y métodos analizando una descripción escrita y asignando nombres a objetos, y verbos a clases o métodos. Este método de nombre y verbo es un comienzo razonable, pero oculta parte de la complejidad de la definición de clases. Con frecuencia se necesitan clases para operaciones en la parte verbal del problema. Por ejemplo, las descripciones de los compiladores constan sobre todo de verbos que describen el

funcionamiento del compilador sobre un programa origen. Pero, al ser tan compleja la compilación, es bueno identificar objetos independientes del compilador, por ejemplo, un analizador sintáctico (parser) y un generador de código. La subdivisión de la compilación en clases demuestra que un procedimiento se puede realizar como método dentro de una clase o como una clase independiente.

Incluso con el método directo de Booch para encontrar clases, resulta difícil conseguir un resultado de alta calidad. Sobre todo, es difícil lograr abstracciones útiles a partir de la definición del problema. Una abstracción útil es el resultado de una estructuración inteligente de la descripción del problema en elementos independientes e intuitivamente correctos. Las abstracciones útiles son normalmente creadas por programadores obsesionados por la simplicidad y con paciencia para volver a trabajar el diseño varias veces hasta que las clases son fáciles de comprender y de especializar. Incluso más raros son los programadores que pueden subir un peldaño más y convertir soluciones específicas para una aplicación en elementos de una librería genérica. Las bibliotecas de clases preexistentes mitigan algunos de estos retos del diseño.

La mayoría de los primeros usuarios de lenguajes y herramientas orientados a objetos han dependido de combinaciones de metodologías basadas en papel y experiencias publicadas por expertos.

No existen reglas estrictas para identificar las clases, pero diseñadores experimentados han ofrecido diferentes reglas básicas. Existe un acuerdo general en que las directrices siguientes ayudan a identificar y definir clases y métodos.

- Modelar con clases las entidades que ocurren de forma natural en el problema.
- Diseñar métodos de finalidad única.
- Diseñar un nuevo método al encontrar una oportunidad de ampliar uno existente.
- Evitar métodos extensos.
- Guardar como variables caso o instancia (modelo) los datos necesitados por

más de un método, o por una subclase.

- El diseñador debe trabajar para la biblioteca de clases, no solo para él mismo, ni para su aplicación actual.

Los diseñadores también han identificado errores cometidos normalmente por novatos en esta fase inicial de diseño de los sistemas orientados a objetos. Los errores comunes incluyen la creación de clases innecesarias y la declaración de clases que no lo son en absoluto. Se insta a los diseñadores a ajustarse a la regla de que cada clase debe corresponder a una abstracción de datos significativa. Existen muchas clases en potencia (es decir, muchos nombres); las innecesarias no son fundamentales para la solución del problema en cuestión. Un método sencillo o rutina no es un buen candidato para una clase. Al contrario que una rutina, que realiza una tarea determinada, una clase debe ofrecer una serie de servicios a objetos de un tipo determinado (Meyer 1989).

Johnson y Foote (1989) estudian la cuestión de cuando crear una nueva clase y cuando añadir un método a una clase existente. Aunque advierten que su criterio no es absoluto, sugieren que se cree una nueva clase cuando:

- Dicha nueva clase represente una abstracción significativa del problema,
- Sea probable que los servicios que proporciona sean utilizados por varias clases más,
- Su conducta sea inherentemente compleja,
- La clase o método haga poco uso de las representaciones de sus valores matemáticos, y
- Si se representara como un método de otra clase, pocos usuarios de ésta lo invocarían.

El ejemplo de compilador mencionado anteriormente ilustra el dilema de realizar una conducta como clase en lugar de como método. El analizador sintáctico del compilador podría muy bien ser modelado como una clase o como un método. Y, considerando que dicho analizador sintáctico sea un método, puede resultar difícil decidir qué clase debe contener dicho método.

Las bibliotecas de clases predefinidas simplifican mucho el proceso de diseño y ayudan a evitar muchas de las dificultades al determinar lo que se debe realizar como clases y como métodos. Los diseñadores siempre deben ver las clases preexistentes en primer lugar, tanto la suministradas con los lenguajes y los entomos de desarrollo como las creadas por el programador. Este consejo se da con la advertencia de que siempre se debe dedicar tiempo adicional de diseño a generalizar y robustecer las clases preexistentes.

Pun y Hilden (1989) advierten que se deben incorporar en el diseño dos estratos (capas) de identificación de objetos, al menos en el nivel conceptual. Estas dos capas o niveles incluyen los objetos de aplicación y los objetos del interfaz del usuario. Los objetos de aplicación son aquellos que no son transparentes para el usuario.

IV. Metodología orientada a objetos.

IV.1 Introducción.

Se presenta aquí una metodología para el desarrollo orientado a objetos y una notación gráfica para representar conceptos orientados a objetos. La metodología consiste en construir un modelo de un dominio de aplicación añadiéndosele detalles de implementación durante el diseño de un sistema. Esta aproximación se denomina la Técnica de Modelado de Objetos (OMT). Dicha metodología consta de las siguientes fases:

IV.1.1 Análisis:

Comenzando desde la descripción del problema el analista construye un modelo de la situación del mundo real que muestra sus propiedades importantes. Dicho analista debe trabajar con quien hace la solicitud para comprender el problema más porque las definiciones del mismo no suelen ser completas ni correctas. El modelo de análisis es una abstracción resumida y precisa de lo que debe hacer el sistema deseado y no de laforma en que se hará. Los objetos del modelo deberán ser

conceptos del dominio de la aplicación, y no conceptos de implementación de la computadora tales como estructuras de datos. Un buen modelo podrá ser comprendido y criticado por expertos de la aplicación que no sean programadores. El modelo de análisis no deberá contener ninguna decisión de implementación. Por ejemplo, una clase Ventana en un sistema de ventanas para una estación de trabajo se describirá en términos de los atributos y operaciones que serán visibles para el usuario.

IV.1.2. Diseño del sistema:

El diseñador de sistemas toma decisiones de alto nivel acerca de la arquitectura global. Durante el diseño, el sistema de destino se organiza en subsistemas basados tanto en la estructura del análisis como en la arquitectura propuesta. El diseñador de sistemas deberá decidir qué características de rendimiento hay que optimar. Seleccionando una estrategia para atacar el problema y efectuando unas reservas de recursos tentativas. Por ejemplo, el diseñador del sistema podría decidir que los cambios habidos en la pantalla de la estación de trabajo sean rápidos y suaves aunque se muevan o borren las ventanas y seleccionará un protocolo de comunicaciones adecuado así como una estrategia de reserva de memoria.

IV.1.3. Diseño de objetos:

El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis que lleven incorporados detalles de implementación. El diseñador añade detalles al modelo de acuerdo con la estrategia establecida durante el diseño del sistema. El foco de atención del diseño de objetos son las estructuras de datos y los algoritmos necesarios para imple-mentar cada una de las clases. Las clases de objetos procedentes del análisis siguen siendo significativas pero se aumentan con estructuras de datos y algoritmos del dominio de la computadora seleccionados para optimar medidas importantes de rendimiento. Tanto los objetos del dominio de la aplicación como los objetos del dominio de la computadora se describen utilizando unos mismos conceptos y una misma notación orientados a objetos aun cuando existan en planos concep-tuales diferentes. Por ejemplo, las operaciones de la clase

Ventana se especifican ahora en términos del hardware y del sistema operativo subyacentes.

IV.1.4 Implementación:

Las clases de objetos y las relaciones desarrolladas durante su diseño se traducen finalmente a un lenguaje de programación concreto, a una base de datos o a una implementación en hardware. La programación debería ser una parte relativamente pequeña del ciclo de desarrollo y fundamentalmente mecánica porque todas las decisiones importantes deberán hacerse durante el diseño. El lenguaje de destino influye en cierta medida sobre las decisiones de diseño pero éste no debería depender de la estructura final de un lenguaje de programación. Durante la implementación es importante respetar las buenas ideas de la ingeniería del software, de tal manera que el seguimiento hasta el diseño sea sencillo y de tal forma que el sistema implementado siga siendo flexible y extensible. Por ejemplo, la clase Ventana se codificaría en un lenguaje de programación utilizando llamadas al sistema de gráficos subyacente en la estación de trabajo. La implementación se describe en la tercera parte de acuerdo con el vehículo de destino.

Es posible aplicar conceptos orientados a objetos a lo largo de todo el ciclo de ida de desarrollo del sistema, desde el análisis hasta la implementación pasando por el diseño. Se pueden traspasar las mismas clases de una etapa a otra sin modificar la notación aunque ganarán detalles adicionales de implementación en las etapas posteriores. Siendo el punto de vista de análisis y el de implementación de Ventana correctos tienen propósitos distintos y representan grados de abstracción. Los mismos conceptos orientados a objetos de identidad, clasificación, polimorfismo y herencia son aplicables a todo el ciclo de desarrollo completo.

Algunas clases no forman parte del análisis sino que se presentan como parte del iseño o de la implementación. Por ejemplo, las estructuras de datos tales como árboles, tablas de dispersión y listas enlazadas no suelen estar presentes en el mundo real. se presentan para que presten su apoyo a algoritmos concretos durante el diseño. Estos objetos de estructuras de datos se utilizan para implementar objetos

del mundo real dentro de la computadora y no derivan directamente sus propiedades del mundo real.

IV.2 Tres modelos.

La metodología OMT emplea tres clases de modelos para describir el sistema: el Modelo de Objetos que describe los objetos del sistema y sus relaciones; el Modelo Dinámico que describe las interacciones existentes entre objetos del sistema; y el Modelo Funcional que describe las transformaciones de datos del sistema. Todos los modelos son aplicables en la totalidad de las fases del desarrollo y van adquiriendo detalles de implementación a medida que progresa el desarrollo. Una descripción completa del sistema requiere los tres modelos.

El modelo de objetos describe la estructura estática de los objetos del sistema y también sus relaciones. El modelo de objetos contiene diagramas de objetos. Un diagrama de objetos es un grafo cuyos nodos son clases de objetos y cuyos arcos son relaciones entre clases.

El modelo dinámico describe los aspectos de un sistema que cambian con el tiempo. El modelo dinámico se utiliza para especificar e implementar los aspectos del control del sistema. Los modelos dinámicos contienen diagramas de estados. Un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos son transiciones entre estados causadas por sucesos*.

El modelo funcional describe las transformaciones de valores de datos que ocurren dentro del sistema. El modelo funcional contiene diagramas de flujo de datos. Un diagrama de flujo de datos representa un cálculo. Un diagrama de flujo de datos es un grafo cuyos nodos son procesos y cuyos arcos son flujos de datos.

Los tres modelos son partes ortogonales de la descripción del sistema completo y están enlazados entre sí. Sin embargo, el más importante es el modelo de objetos porque es necesario para describir qué está cambiando o transformándose antes de describir cuándo o cómo cambia.

IV.3 La Tecnica de Modelado de Objetos.

Resulta útil modelar un sistema desde tres puntos de vista distintos, aunque relacionados, cada uno de los cuales captura aspectos importantes del sistema, pero siendo todos ellos necesarios para una descripción completa. La Técnica de Modelado de Objetos (OMT) es el nombre que se da a una metodología que combina estos tres puntos de vista para el modelado de sistemas. El modelo de objetos representa los aspectos estáticos, estructurales "de datos" del sistema. El modelo dinámico representa los aspectos temporales, de comportamiento "de control" del sistema. El modelo funcional representa los aspectos transformacionales "de función" del sistema. Un procedimiento típico de software contiene esos tres aspectos: utiliza estructuras de datos (modelo de objetos) secuencia las operaciones en el tiempo (modelo dinámico) y transforma valores (modelo funcional). Cada modelo contiene referencias a entidades de otros modelos. Por ejemplo, las operaciones se asocian a los objetos en el modelo de objetos pero se expanden de forma más completa en el modelo funcional.

Las tres clases de modelos desglosan el sistema en puntos de vista ortogonales que se pueden representar y manipular empleando una notación uniforme. Los distintos modelos no son completamente independientes —un sistema es algo más que una colección de partes independientes— pero cada modelo puede ser examinado y comprendido por sí mismo en gran parte. Las interconexiones entre los distintos modelos son limitadas y explícitas. Por supuesto, siempre es posible crear diseños malos en los cuales los tres modelos estén tan entremezclados que no sea posible separarlos, pero los buenos diseños aíslan los distintos aspectos del sistema y limitan el acoplamiento entre ellos.

Cada uno de los tres modelos va evolucionando durante el ciclo de desarrollo. Durante el análisis un modelo del dominio de la aplicación se construye sin tener en cuenta la implementación que se efectuará eventualmente. Durante el diseño se añaden al modelo estructuras del dominio de la solución. Durante la implementación se codifican, tanto las estructuras del dominio de la aplicación como las estructuras

del dominio de la solución. La palabra modelo tiene dos dimensiones —una vista de un sistema (modelo de objetos, modelo dinámico o modelo funcional) y una fase de desarrollo (análisis, diseño o implementadion). El significado suele estar claro a partir del contexto.

IV.3.1 El modelo de objetos.

El modelo de objetos describe la estructura de los objetos de un sistema —identidad, relaciones con otros objetos, atributos, y operaciones—. El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. Los cambios y transformaciones carecen de sentido a no ser que haya algo que se pueda cambiar o transformar. Los objetos son las unidades en las que dividimos el mundo, las moléculas de nuestros modelos.

Nuestro objetivo al construir un modelo de objetos es capturar aquellos conceptos del mundo real que sean importantes para una aplicación. Al modelar un problema de ingeniería, el modelo de objetos debería contener términos familiares para los ingenieros; al modelar un problema comercial, términos del mundo de los negocios; al modelar una interface de usuario, términos del dominio de la aplicación. Un modelo de análisis no debería contener estructuras de computadora a no ser que la aplicación que se esté modelando sea inherentemente un problema de computadoras, tal como un compilador o un sistema operativo. El modelo de diseño describe la forma de resolver el problema y puede contener estructuras de computadora.

El modelo de objetos se representa gráficamente mediante diagramas de los mismos que contenga clases de objetos, organizándose éstas en jerarquías que comparten una estructura y comportamiento comunes y que estén asociadas con otras clases. Éstas definen los valores de los atributos que lleva cada instancia de objeto y las operaciones que efectúa o sufre cada uno.

IV.3.2 El modelo dinámico.

El modelo dinámico describe aquellos aspectos del sistema que tratan de la temporización y secuencia de operaciones —sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos—. y la organización de sucesos y de estados. El modelo dinámico captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en la que estén implementadas.

El modelo dinámico se representa gráficamente mediante diagramas de estado. Cada uno de estos diagramas muestra el estado y las secuencias de sucesos que son admisibles en un sistema para una clase de objetos. Los diagramas de estado también hacen alusión a los demás modelos. Las acciones de los diagramas de estado se corresponden con funciones procedentes del modelo funcional; los sucesos de un diagrama de estado pasan a ser operaciones que se aplican a objetos dentro del modelo de objetos.

IV.3.4 El modelo funcional.

El modelo funcional describe aquellos aspectos del sistema que tratan de las transformaciones de valores —funciones, correspondencias, restricciones y dependencias funcionales. El modelo funcional captura lo que hace el sistema, independientemente de cuando se haga o de la forma en que se haga.

El modelo funcional se representa mediante diagramas de flujo de datos. Estos muestran las dependencias entre los valores y el cálculo de valores de salida a partir de los de entrada y de funciones, sin considerar cuando se ejecutan las funciones, ni siquiera si llegan a ejecutarse. Los conceptos tradicionales de computación tales como árboles de expresiónes (sintácticos) son ejemplos de modelos funcionales, así como lo son unos conceptos menos tradicionales como son las hojas de cálculo. Las funciones se invocan como acciones en el modelo

dinámico y se muestran como operaciones que afectan a objetos en el modelo de objetos.

IV.3.5 Relaciones entre modelos.

Cada modelo describe un aspecto del sistema, pero contiene referencias a los demás modelos. El modelo de objetos describe la estructura de datos sobre la cual operan los modelos dinámico y funcional. Las operaciones del modelo 'de objetos se corresponden con sucesos en el modelo dinámico y con funciones en el modelo funcional. El modelo dinámico describe la estructura de control de los objetos, muestra decisiones que dependen del valor de los mismos y dan lugar a acciones que modifican valores y que invocan a funciones. El modelo funcional describe las funciones pedidas por operaciones en el modelo de objetos y acciones del modelo dinámico. Las funciones operan sobre valores de datos especificados por el modelo de objetos. El modelo funcional también muestra limitaciones de los valores de los objetos.

Ocasionalmente, existen ambigüedades acerca del modelo que debería contener una cierta información. Esto es natural, porque cualquier abstracción no es más que una aproximación burda de la realidad; inevitablemente, siempre hay algo que se encuentra a caballo entre ambos mundos. Algunas propiedades de un sistema pueden ser representadas inadecuadamente mediante los modelos. Esto también es normal porque ninguna abstracción puede capturarlo todo; el objetivo es simplificar la descripción del sistema sin sobrecargar el modelo con tantas estructuras que pase a ser un obstáculo y no una ayuda. Para aquellas cosas que no sean capturadas adecuadamente por el modelo, el lenguaje natural o una notación específica de la aplicación, sigue siendo una herramienta perfectamente admisible.

BIBLIOGRAFIA

Modelado y Diseño Orientados a Objetos.
Rumbaugh James, Blaha Michael, Premerlani William
Eddy Frederick, Lorensen William.
Prentice hall Primera Edición.
1991. España.

Software Orientado a Objetos.
Ann L. Winbland, Samuel Edwards, David R King.
Addison-Wesley/Diaz de SantosPrimera Edición
1993. USA.

Object Oriented Software Engeneering.
Ivar Jacobson, Magnus Christerson, Patrik Jonsson.
Addison-Wesley/ACM Press Primera Edición.
1993 Inglaterra.

Analisis y Diseño Orientado a Objetos.
Grady Booch.
Addison-Wesley/Diaz de SantosSegunda Edición.
1996 USA.