37
2ej

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA PROFESIONAL DE ESTUDIOS PROFESIONALES
"ARAGON"

INGENIERIA EN COMPUTACION

LA METODOLOGIA DE ORIENTACION A OBJETOS COMO NUEVA HERRAMIENTA PARA EL ANALISIS Y DISEÑO DE SOFTWARE.

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE
INGENIEROS EN COMPUTACION
PRESENTAN:

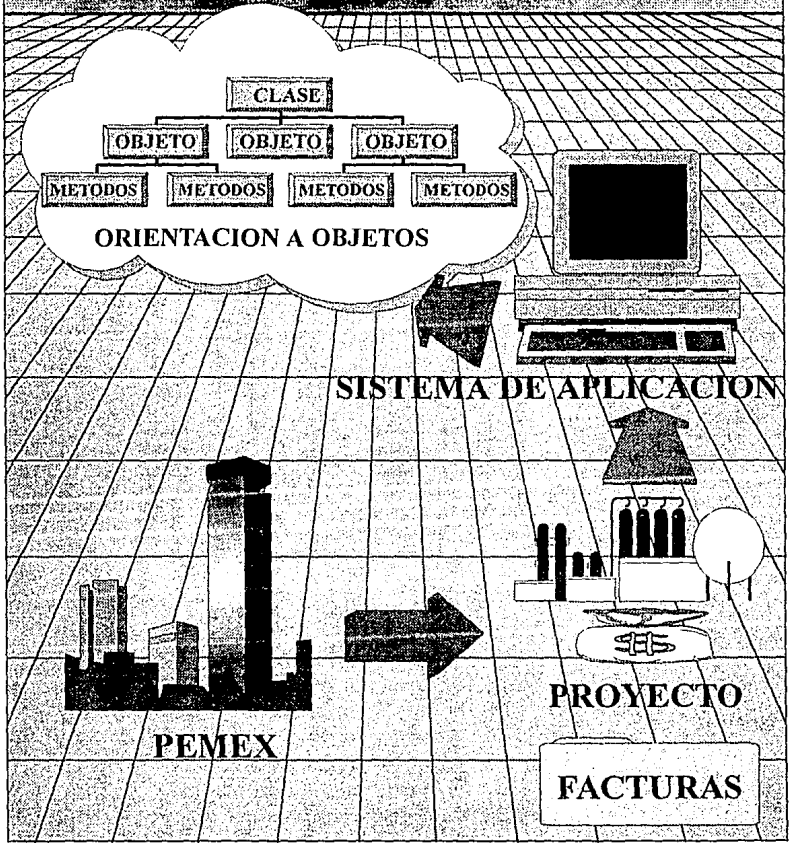
ADRIANA SANTOS ROSAS
MARIA DEL CARMEN SOTO CARIÑO

Asesor de tesis:
Ing. Victor Raúl Velasco Vega

San Juan de Aragón, Estado de México. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

LA METODOLOGIA DE ORIENTACION A OBJETOS COMO NUEVA NUEVA HERRAMIENTA DE ANALISIS Y DISEÑO DE SOFTWARE.



LA METODOLOGIA DE ORIENTACION A OBJETOS COMO NUEVA HERRAMIENTA PARA EL ANALISIS Y DISEÑO DE SOFTWARE.

OBJETIVO:

1.- PROPORCIONAR LOS CONCEPTOS FUNDAMENTALES DE LA ORIENTACION A OBJETOS.

2.- DESCRIBIR EL PROCESO DE ANALISIS Y DISEÑO PROPUESTO POR GRADY BOOCH.

3.- UTILIZAR LA PROGRAMACION ORIENTADA A OBJETOS PARA RESOLVER LA PROBLEMATICA QUE ACTUALMENTE SE ENCUENTRA EN EL DESARROLLO DE SOFTWARE :

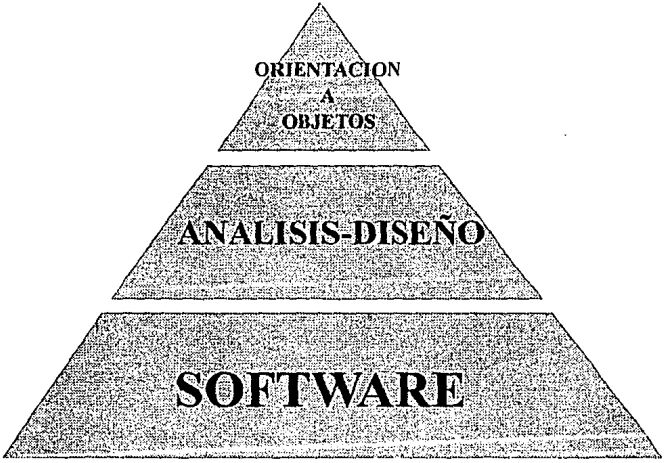
a) COMPLEJIDAD, MANTENIMIENTO Y COSTO.
b) FALTA DE REUTILIZACION.
c) IMPRODUCTIVIDAD DEL PROGRAMADOR.
d) INFLEXIBILIDAD PARA EL CAMBIO.

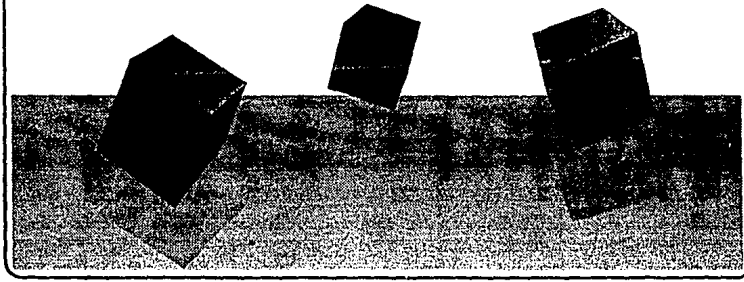
4.- EMPLEAR LA PROGRAMACION ORIENTADA A OBJETOS EN EL DESARROLLO DEL **"SISTEMA DE CONTROL DE FACTURAS Y PROYECTOS"** DE PETROLEOS MEXICANOS; Y COMPARAR LA RENTABILIDAD DE ESTA, CONTRA LA PROGRAMACION TRADICIONAL Y ASI OFRECER SUS VENTAJAS EN LOS ASPECTOS TECNICOS Y COSTO-BENEFICIO, RESOLVIENDO DE ESTA FORMA LOS PROBLEMAS ANTES MENCIONADOS.

5.- OFRECER ESTE DOCUMENTO COMO HERRAMIENTA DE APOYO PARA EL ANALISIS, DISEÑO Y DESARROLLO DE SOFTWARE; ASI COMO LA VISION PANORAMICA DEL ESTADO ACTUAL DE PROGRAMACION ORIENTADA A OBJETOS Y SUS TENDENCIAS FUTURAS.

INDICE

INTRODUCCION





INTRODUCCION

La computadora es el instrumento más complejo que ha inventado el hombre y por muchas décadas permanecerá siendo el más potente. Nuestras vidas se ven constantemente afectadas por las aplicaciones cada vez mayores de las computadoras. En el futuro, servicios utilitarios de computadoras serán tan corrientes como los servicios telefónicos y de energía eléctrica del presente. Debido a la continua investigación y desarrollo de este formidable instrumento y sus aplicaciones, ha surgido un conjunto de conocimientos extensos e impresionantes.

En la breve historia de las computadoras de electrónica digital, los años 50's y los 60's fueron décadas de hardware. Los años 70's fueron un período de transición y un tiempo de reconocimiento del software. Ahora estamos en la década del software. De hecho, los avances de la informática pueden llegar a estar limitados por nuestra incapacidad de producir software de calidad que pueda hacerse con la enorme capacidad de los procesadores de la era de los 80's.

Durante la pasada década se han reconocido cada vez más las circunstancias que colectivamente se señalan como crisis del software. Los costos del software han crecido dramáticamente, convirtiéndose en la parte más costosa en dólares de muchos sistemas basados en computadora. Se fijan agendas y fechas de terminación pero raras veces se cumplen. A medida que crece un sistema de software, la calidad se hace más sospechosa. Los responsables de los proyectos de desarrollo de software disponen de pocos datos históricos que se puedan usar como guía y cada vez menos control sobre el curso de un proyecto.

Como respuesta a la crisis del software ha evolucionado un conjunto de técnicas denominadas colectivamente *Ingeniería de Software*. Estas técnicas se enfrentan con el software como un producto de ingeniería que requiere planificación, análisis, diseño, implementación, prueba y mantenimiento. De tal forma que puede definirse a la Ingeniería de Software como " ***la disciplina tecnológica y administrativa dedicada a la producción sistemática de productos de programación, que son desarrollados y modificados a tiempo dentro de un presupuesto definido***".

Las técnicas primordiales de esta nueva disciplina tecnológica son mejorar la calidad de estos productos y aumentar la productividad y satisfacción profesional de los ingenieros de esta disciplina. La ingeniería de software es una disciplina pragmática que confía en las ciencias de la computación para obtener los fundamentos científicos de la misma manera que las ramas de la ingeniería tradicional, como las ingenierías eléctrica y química se valen de la física y la química. Ya que la ingeniería de software se preocupa del desarrollo y mantenimiento de productos de la tecnología moderna, es necesario utilizar técnicas de resolución de problemas comunes a todas las ramas de la ingeniería; estas técnicas sientan las bases de la planeación y administración de proyectos, análisis de sistemas, diseño metódico, fabricación cuidadosa, validación profusa y mantenimiento continuo del producto. Para efectuar esto se requiere de la aplicación de una notación adecuada, así como de herramientas y técnicas en cada área; además, los ingenieros deben equilibrar en forma práctica los principios básicos con los aspectos económicos y las preocupaciones sociales cuando resuelven problemas y desarrollan productos tecnológicos.

Los conceptos de las ciencias de la computación y administración de la economía y de la comunicación están combinados dentro del marco de la resolución de problemas; el producto de esto recibe el nombre de ingeniería de software. Dentro de esta rama existen técnicas y herramientas las cuales suelen ser un tanto obsoletas y poco confiables en relación al avance tecnológico de las más recientes. Por ello que actualmente ha salido al mercado una nueva herramienta que vislumbra un panorama para la generación de software diferente a lo convencional que es la llamada **Orientación a Objetos** que es empleada en las diversas ramas de la ingeniería de software ya sea como metodología de análisis y diseño, como un tipo de programación o empleada en los lenguajes orientados a objetos y puede constituir una de las herramientas más modernas y poderosas para construir sistemas de información, interfaces hombre máquina y software de aplicación para las más diversas actividades.

Existen razones para pensar que esta herramienta es una de las innovaciones más importantes que se han dado en el desarrollo del software desde la introducción de los superlenguajes de programación en la década de los 60's, o de las bases de datos una década después.

No sólo se trata de una nueva forma de entender y hacer computación, sino la posibilidad real de reducir los costos del desarrollo del software, y al mismo tiempo, de incrementar la capacidad técnica para construir sistemas de gran complejidad interna y de operación amigable.

Es por ello que hoy en día es sumamente interesante descubrir los fenómenos que hacen conocer a esta herramienta como un poderoso instrumento, para la solución de muchos problemas en la **Ingeniería de Software**.

En el **capítulo I**, se describen los fundamentos de la orientación a objetos (conceptos, definiciones de diferentes autores) además se observa la visión que alcanza hasta el momento esta innovación de software, y se introduce al lector de manera general y sencilla al entendimiento de los conceptos más elementales (**Orientación a Objetos**).

En el **capítulo II**, se presenta una metodología de análisis y diseño clásica y la metodología orientada a objetos (en base al autor **Grady Booch**). Se describen los conceptos básicos de la nueva metodología así como la notación empleada para la misma, además de hacer un breve análisis comparativo sobre la metodología clásica y la de **Grady Booch**.

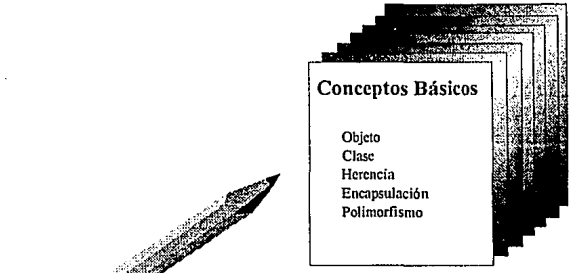
En el **capítulo III** llamado **"Programación y Lenguajes"**, se da un antecedente (características) de los métodos de programación más populares y usados en la actualidad, además de incluir la definición y características (elementos) de la programación orientada a objetos en los diferentes lenguajes de programación; se presentan, además algunos Lenguajes y Bases de Datos que existen en la actualidad Orientados a Objetos, dando sus aspectos generales de los mismos. En este capítulo se hace mención de costos, plataformas de software y paquetes de software que actualmente se encuentran en el mercado.

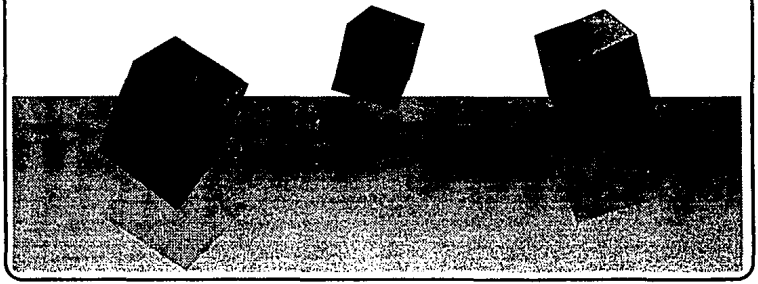
En el **capítulo IV**, se presenta la aplicación y comprobación de todos aspectos mencionados en los anteriores capítulos de este documento, debido a que aquí se muestra la realización del Sistema de Aplicación de Software llamado Control de Facturas y Proyectos (**CONFYP**), donde se describe al lector desde el análisis, diseño, elección del paquete de software empleado, y el desarrollo del sistema finalizando con la descripción del manejo de sistema ya construido.

En el **capítulo V**, se presenta una visión panorámica de la situación de la Orientación a Objetos en nuestro país, las perspectivas dentro del mercado, la problemática que existe dentro de la comunidad del software para la aceptación de esta innovación (costos, capacitación, mentalidad). Además se muestra las ventajas y desventajas de esta innovación.

CAPITULO I

Fundamentos





FUNDAMENTOS

La problemática cuya resolución está encomendada a la programación conocida como ingeniería de software suele ser de una enorme complejidad. Puede ser muy difícil comprender la naturaleza del problema sobre todo si el sistema que sirve como modelo para el software que va a desarrollarse es nuevo y no está automatizado. Debido a esto nace la necesidad de crear una nueva visión de entender, interpretar y desarrollar software; que en la actualidad se le denomina ***"Orientado a Objetos"***.

Las palabras ***"Object-Oriented"***, o dicho en español, **Orientado a Objetos**, se han puesto muy de moda en los últimos años en el ámbito de la computación. ***Es toda una nueva cultura que pretende desplazar el diseño estructurado utilizado tradicionalmente; sobre todo, auxiliar efectivamente el desarrollo de sistemas de software complejo.***

Las técnicas de software orientadas a objetos han generado un amplio interés en los últimos años. Por ello hoy en día el mercado de software para desarrollo de aplicaciones está inundado de productos, todos ellos con fuertes campañas publicitarias que son anunciadas como **Orientado a Objetos.** Razón por la cual es necesario conocer los conceptos básicos (**Fundamentos**) de éste término.

En este capítulo se dará un breve antecedente de la evolución histórica de esta innovación de software, además de mostrar una definición del término así como sus características propias y definiciones de los conceptos fundamentales utilizados para este término.

LENGUAJE	SIGLAS
ADA	Hace referencia al nombre de la primera programadora (Augusta ADA Byron).
ALGOL-60	Lenguaje Algorítmico.
BASIC	Beginner's All-Purpose Symbolic Instruction Code.
CLOS	Common Lisp Object System.
COBOL	Lenguaje Común Orientado a Negocios.
EIFFEL	Hace referencia a la Torre de EIFFEL de París, porque el diseñador de este lenguaje era Francés.
FORTRAN	FORmula TRANslation
LISP	Lenguaje de Proceso de Listas.
PASCAL	Hace referencia al famoso matemático francés del siglo XVII, Blaise PASCAL.
SIMULA-67	Viene de la palabra de Simulación.
SMALLTALK	Indica un lenguaje de pocas palabras.

Figura 1.1 Tabla de referencia de siglas y significado de lenguajes

Nota: Es necesario decir que en la actualidad existen infinidad de lenguajes y bases de datos orientados a objetos, pero en este documento solo se mencionan algunos como los descritos anteriormente en la figura 1.1.

1.1 EVOLUCION

El desarrollo acelerado de la ingeniería de software dio como resultado el surgimiento de una nueva herramienta de diseño, la llamada *"Orientación a Objetos"*.

Esta innovadora herramienta toma como base para su desarrollo y aplicación la evolución histórica de los diferentes lenguajes y métodos de programación.

Uno de los lenguajes que propició el desarrollo de la metodología de la programación estructurada es el conocido ALGOL-60, resultado de un grupo formado por investigadores reconocidos a nivel mundial. ALGOL-60 marcó el inicio de una nueva generación de lenguajes de programación estructurados.

El precursor de los lenguajes orientados a objetos, SIMULA-67, nació en Noruega, donde un grupo de investigadores, encabezados por Dahl y Nygaard, buscaba modificar a los lenguajes ya existentes, diseñados para aplicaciones numéricas, con el objetivo de hacerlos aptos para programar las simulaciones discretas de los problemas del mundo real. Los problemas a simular se componían de objetos que cooperaban o interactuaban de forma variada entre sí. Los noruegos extendieron a ALGOL-60 con los conceptos de objetos, clases de objetos y jerarquías de herencias entre clases, que caracterizan hoy en día la POO (Programación Orientada a Objetos).

Varios años después, las ideas nacidas en SIMULA se adoptaron en el diseño e implantación del lenguaje SMALLTALK. SMALLTALK fue desarrollado a finales de los 70's por un grupo de los Laboratorios Xeros, de Palo Alto, CA, en los Estados Unidos, bajo la dirección de Andrew Kay, en el proyecto conocido como "Dynabook", cuyo objetivo fue crear una interfaz muy amigable para una computadora personal. El lenguaje SMALLTALK resultado de este esfuerzo, retomó los conceptos de objeto, clase y herencia de SIMULA, y los conjugó con la flexibilidad de la programación interactiva y libre de tipificación del lenguaje LISP. Al popularizarse SMALLTALK, se impuso una nueva terminología, apareció el término (Orientación a Objetos) y empezó el "boom" de la Programación Orientada a Objetos.

Es interesante notar que no fue sino una década después del nacimiento de SIMULA que los conceptos presentes en el lenguaje tuvieron un amplio impacto.

Durante la década en la que programación orientada a objetos estuvo en estado latente, la cultura computacional tuvo grandes avances importantes, en particular en las áreas de software de base (compiladores, sistemas operativos, etc.).

La segunda mitad de los 80's y lo que se lleva de los 90's, se han distinguido por una gran cantidad de propuestas de nuevos lenguajes con programación orientada a objetos. Entre los más importantes, se encuentra la extensión del lenguaje C, conocida como C++ de Stroustrup (el más apegado a la herencia de SIMULA), EIFFEL de Meyer; y una extensión orientada a objetos de LISP, conocida como CLOS. También cabe hacer resaltar las distintas modificaciones de PASCAL, como OBJECT PASCAL o recientes versiones de TURBO PASCAL.

Actualmente, las ideas originarias en los lenguajes de programación están siendo usadas en distintas áreas de la computación como son las bases de datos orientadas a objetos.

A continuación se muestra un cuadro sinóptico donde se describe brevemente la historia de algunos de los lenguajes orientados a objetos ver figura 1.2.

AÑO	SUCESO
1967	Kristen Nvgaard y Ole-Johan Daul desarrollan SIMULA 67.
1970-1972	Dan Engalls escribe el evaluador de SMALLTALK. Alan Kay desarrolla SMALLTALK 72.
1977	Se desarrollan en el MIT extensiones basadas en Objetos al Lenguaje LISP.
1980	Xeros define SMALLTALK 80, dona licencias a Apple, DEC, HP, y Tektroenic.
1980	Bjarne Stroustrup en Bell Laboratorios en Murray Hill, New Jersey, añade varias extensiones al lenguaje C y los llama "C con clases".
1983	O-O PASCAL es desarrollado.
1983	"C con clases" cambia de nombre a C++.
1984	Object PASCAL es definido. Objetive C es creado por Brad Cox .
1985	C++ es creado por Bjarne Stroustrup en AT&T.
1988	Se forma "Object Tecnology International".
1989	Apple libera C++ para Mac, Borland agrega Objetos a Turbo PASCAL , AT&T libera C++ 2.0.
1990	Borland lanza Turbo C++.
1992	Microsoft lanza C/C++.

Figura 1.2 Evolución de la orientación a objetos

1.2 DEFINICION

La orientación a objetos como una nueva herramienta de software para la creación de sistemas, se puede definir como:

La orientación a objetos es un modelo de software y desarrollo (ingeniería) de disciplinas que hacen más fácil la construcción de sistemas complejos a partir de componentes individuales. [1]

Intuitivamente, el atractivo de la orientación a objetos es el de mejorar las herramientas y conceptos con los cuales se pueda modelar y representar el mundo real tan fielmente como sea posible.

Como se muestra en la figura 1.3, cuando se usan técnicas convencionales, el código generado por un problema del mundo real, se obtiene primero codificando el problema y posteriormente transformarlo en términos del lenguaje de computadora de Von Neuman.

MODELO DE VON NEUMANN

La idea central de este modelo es almacenar las instrucciones del programa de una computadora en su propia memoria, logrando así que la máquina siga los pasos definidos por su *"programa almacenado"*. Para ello es necesario saber como comunicarle a la computadora qué operaciones efectuar sobre los datos previamente almacenados en la memoria, considerando que la función de la memoria es de almacenar los datos.

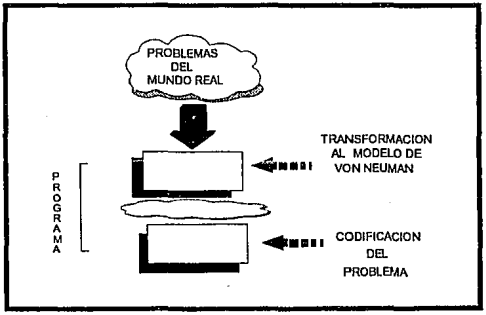


Figura 1.3 Programación convencional

Los conceptos y herramientas orientados a objetos están habilitando las tecnologías que permiten que los problemas del mundo real se expresen fácil y naturalmente.

[1] Khoshafian Setrag; "Object Orientation, Concepts, Languages, Databases, User Interfaces", Editorial Wiley, 1990; pp.6

1.3 CARACTERISTICAS

Como toda innovación, la orientación a objetos presenta características que la hacen única y que en el caso de sistemas complejos facilitan su construcción, algunas de ellas se mencionan a continuación:

1. **La orientación a objetos provee mejores métodos para construir complejos sistemas de software a partir de unidades modularizadas y reutilizables.**

2. **Fácil interacción con el ambiente de la computación usando metáforas familiares.**

3. **Construcción de componentes de software reutilizables y librerías de fácil extensión.**

La orientación a objetos se esfuerza por satisfacer las necesidades de los usuarios, así como el desarrollo de productos de software.

1.4 OBJETOS

Un objeto es un componente del mundo real que se transforma en el dominio del software.

Para entender la definición de objeto se puede emplear el recurso de compararlo con algo que resulta familiar. Para nuestros propósitos, se puede comparar a un objeto con una célula ya que ésta es la unidad básica con la cual todos los seres vivos están compuestos, es un *"paquete"* orgánico, que como los objetos, combina información y comportamiento. Las células son envueltas por una membrana que sólo permite ciertos tipos de intercambios químicos con otras células. Todas las interacciones entre células se dan a través de *MENSAJES* químicos reconocidos por la membrana de la célula y pasados hacia el interior de ella. De forma semejante los objetos se comunican entre ellos a través de mensajes, los cuales determinan su comportamiento.

Una de las causas por lo cual se pretende aplicar la noción de objeto al campo de la informática es reducir los tiempos de desarrollo de software.

En el contexto de un sistema basado en computadora, un objeto es normalmente un procedimiento o consumidor de información o un elemento de información. Por ejemplo, objetos

típicos pueden ser: máquinas, archivos, visualizaciones, conmutadores, señales, cadenas alfanuméricas o cualquier persona, lugar o cosa.

Cuando un objeto se transforma en una realización software, consta de una estructura de datos y procesos. A los procesos se les llama ***operaciones*** o ***métodos***, que pueden transformar legítimamente la estructura de los datos. Las operaciones contienen el control y las construcciones procedimentales que pueden ser llamadas mediante un mensaje (una petición al objeto) para que ejecute una de sus operaciones.

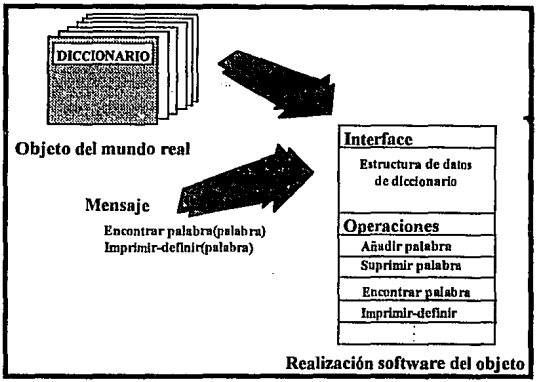


Figura 1.4 Objetos

Un objeto del mundo real un diccionario se transforma en una realización software para un sistema basado en computadora (ver figura 1.4). La realización software de un diccionario exhibe una estructura de datos privada y las operaciones relativas a ellas. Las entradas del diccionario están compuestas de palabras, una guía de pronunciación y una o más definiciones. El diccionario de objetos también contiene un conjunto de operaciones (por ejemplo añadir-palabra, encontrar-palabra) que puede procesar los elementos de la estructura de datos descrita anteriormente. La parte privada de un objeto es la estructura de datos y el conjunto de operaciones para la estructura de datos. De tal manera que se puede definir a un objeto como:

Una entidad abstracta caracterizada por un estado interno, su comportamiento y una entidad propia.

Así, un objeto se puede considerar como una entidad que agrupa a estructuras de datos con el conjunto de operaciones que las manipulan. Las estructuras de datos, conocidas también como variables de instancia, conforman el estado interno de los objetos. El valor de las variables puede cambiarse dinámicamente durante la vida de un objeto. Las operaciones que un objeto es capaz de realizar determinan su comportamiento.

1.4.1 Mensaje

Un mensaje es el medio de comunicación con un objeto, el cual determina su comportamiento.

Un objeto también tiene una parte compartida que es su interfaz. Los mensajes se mueven a través de la interfaz y especifican que operaciones del objeto se desean, pero no como van a realizar la operación. El objeto que recibe un mensaje determina cómo se implementa la operación solicitada.

Los objetos se comunican entre si a través de mensajes. Un mensaje es simplemente el nombre de un objeto receptor junto con el nombre de uno de sus métodos. Un mensaje es una solicitud para el objeto receptor para llevar a cabo el método indicado y regresar el resultado de la acción.

De lo anterior se deduce que, aunque la técnica del envío de mensajes es muy sencilla, mejora notablemente el desarrollo de software en grupos de trabajo, ya que el programador que utiliza el objeto tan solo se preocupa de enviar el mensaje, y le corresponde al programador encargado de desarrollar el objeto, el asegurar que el mensaje va a ser bien interpretado.

1.4.1.1 Partes que constituyen un mensaje

1. Nombre del objeto receptor.
2. Nombre del METODO que el objeto receptor sabe como ejecutar.
3. Parámetros que este objeto requiere para realizar sus funciones. Esta parte es opcional, si el método no necesita información adicional, no hay parámetros en el mensaje.

1.4.2 Protocolo

Es el conjunto de mensajes a los cuales responden un objeto. El turno de la respuesta es determinado por la clase a la que pertenece el objeto. El protocolo de un objeto describe la interfaz. Esto es la colección de métodos definida en la instancia de clases.

La descripción del protocolo no es más que un conjunto de mensajes y un comentario correspondiente para cada mensaje.

1.4.3 Otros autores

Existen diversas opiniones sobre el concepto objeto, razón por la cual cada autor da su propio enfoque sobre este término. En la figura 1.5 se muestra definiciones de objeto dadas por diferentes autores.

AUTOR	DEFINICION
BRAD COX	"Los objetos están formados por datos privados y por un conjunto de operaciones que pueden accesar estos datos. A los objetos se les pide realizar una de sus operaciones mandándoles un mensaje."[2]
PETER WEGNER	"Los objetos son entidades autónomas que responden a mensajes u operaciones y comparten un estado."[3]
S. DANFORTHYC. TOMLINSON	"En la programación orientada a objetos, los objetos cuentan con una identidad denominada *Self*, en ésta persisten el tiempo independientemente de los cambios en el estado del objeto. Los objetos son inteligentes y responden a llamados dirigidos a ellos (mensajes). La forma que un objeto va interpretar un mensaje se denomina método."[4]
GRADY BOOCH	"Un objeto es algo que tiene su estado, comportamiento e identidad. La estructura y comportamiento de los objetos similares se definen en su clase común".[5]

Figura 1.5 Definiciones de otros autores

1.4.4 Características

Los objetos se utilizan para llevar entes (es decir, lo que existe o puede existir) y fenómenos pertenecientes al mundo real a la computadora utilizando algunas herramientas. Los elementos que conforman a un objeto se muestra en la figura 1.6.

---

[2] Swaine, Michael, "Babbit's Guide to OPP", Dr. Dobb's journal, Vol.14 , No 6, junio, 1989.
[3] Wegner, Peter, "Dimensions of Objet-based languaje desingn", OOPSLA'87, Conference roceedings, SIGPLAN Notices, Vol. 22, No 12, diciemmbre, 1987.
[4] Danforth, Scott y tomlinson, Chris, "Type Theories and Objet-Oriented Programming", ACM computing Surveys, Vol. 20, No. 1, marzo, 1988.
[5] Grady Boch, "Object-Oriented Desing with Applications", The Benjamin/Cummings Pub. Comp, 1993.

1. **ESTADO**
2. **COMPORTAMIENTO**
3. **IDENTIDAD**

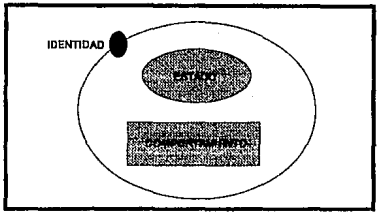


Figura 1.6 Elementos que conforman un objeto

1. **Estado:** de un objeto lo conforman sus propiedades, es decir, una característica, un rasgo o una cualidad inherente (esto es, que por su naturaleza están unidos inseparablemente con otra cosa) que hace de un objeto algo único.

2. **Comportamiento:** de un objeto está definido por la forma en que actúa y reacciona. El comportamiento de un objeto esta definido por las *operaciones* que pueden ser invocadas para el objeto.

En los lenguajes de programación las *operaciones* asociadas a un objeto se conocen como sus *métodos* (SMALLTALK) o funciones miembro (C++).

Existen diferentes ***tipos de métodos.***

A) Modificadores	(modifican el estado de un objeto)
B) Selectores	(permiten ver el estado de un objeto sin alterarlo)
C) Iterativos	(permiten acceso a todas partes del objeto en cierto orden)
C) Constructores	(crean e inicializan un objeto)
E) Destructores	(destruyen a un objeto liberando la memoria que ocupa)

3. **Identidad:** es la propiedad de un objeto que lo distingue de todos los demás objetos. Puede haber varias variables que denotan el mismo objeto, pero no puede haber varios objetos con la misma identidad.

1.5 MODELO DE OBJETOS

Las técnicas de análisis y diseño orientadas a objetos son muy variadas, pero existen cuatro elementos básicos para crear un modelo como se muestra en la figura 1.7, las cuales se encuentran presentes en todas ellas, además de otros tres elementos.

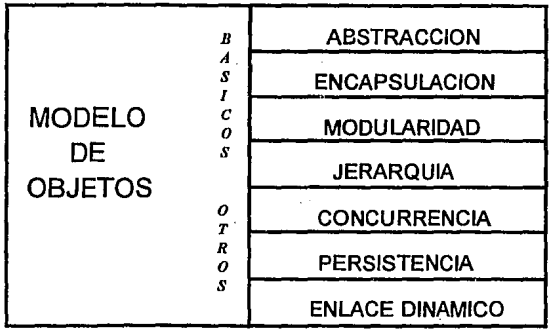


Figura 1.7 Elementos del modelo de objetos

1. **Abstracción:** es una forma de caracterizar a los objetos fijándose solamente en propiedades que nos interesan en un momento dado, e ignorando otros detalles que no son importantes. Las características escogidas son relativas a la perspectiva del observador. La idea de abstracción de datos se utiliza para crear un objeto que es una entidad que agrupa en él mismo los datos y las operaciones que afectan a esos datos, o lo que es igual, las funciones que operan sobre esos datos.

Datos:	Información contenida en el objeto.
Funciones:	Comportamiento de los datos.

Cuando se genera el modelo conceptual de un sistema, se efectúa un proceso de abstracción sobre las entidades u objetos del mundo real que se desea modelar. Cada abstracción como ya

se mencionó, se asocia con una serie de características estructurales y de comportamiento, que son relevantes para el modelo (figura 1.8). Las metodologías orientadas a objetos permiten la asociación de una abstracción del mundo real con un componente computacional que refleja directamente las características de la abstracción.

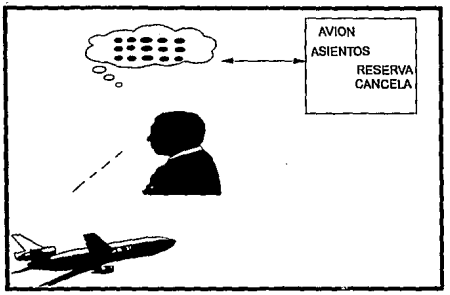


Figura 1.8 Abstracción

2. **Encapsulación:** es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales (ver figura 1.9).
   La encapsulación hace que el objeto tenga un *interfaz público* y una *representación privada*; este principio se conoce como *ocultación de la información*.

La interfaz de un componente cumple con una doble tarea, por un lado especifica los servicios que ofrece un componente y los requerimientos para su correcta ejecución y, por el otro, oculta todos aquellos que no se encuentren explícitamente en la interfaz.

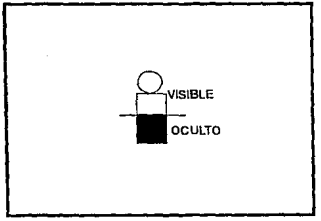


Figura 1.9 Encapsulación

3. **Modularidad:** un módulo es una agrupación de abstracciones lógicamente relacionadas. La descomposición de un sistema en módulos debe efectuarse bajo una metodología que facilite la producción de componentes que puedan ser fácil de combinar con otros, bajo reglas específicas, para producir un sistema (ver figura 1.10).

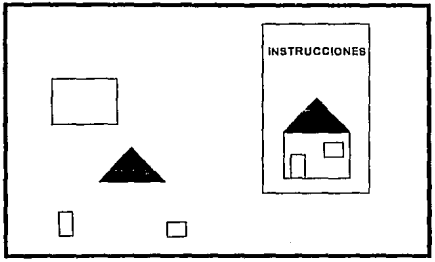


Figura 1.10 Modularidad

4. **Jerarquía:** es la manera de organizar y ordenar a las abstracciones (ver figura 1.11). Las jerarquías se manejan por estructura (objeto) y por especialización (clase).
   La jerarquía por estructura se conoce como la agregación, mientras que la de especialización como herencia sencilla o múltiple.
   La clasificación ordena al conocimiento en jerarquías, agrupando cosas que tienen una estructura común o comportamiento similar.

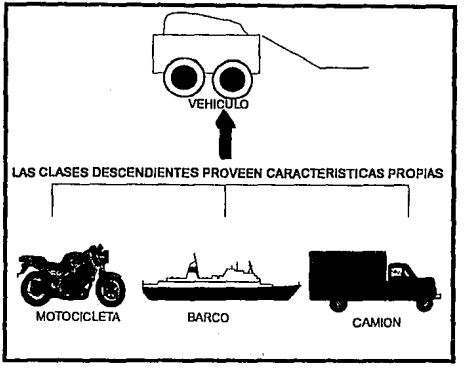


Figura 1.11 Jerarquía

1.5.1 Otros conceptos en el modelo de objetos

5. **Concurrencia:** es la posibilidad de tener objetos activos y pasivos en un sistema orientado a objetos. Este concepto no es necesario para hablar del modelo orientado a objetos pero es muy útil porque varios sistemas son esencialmente concurrentes.

6. **Enlace Dinámico:** también conocido como enlace retardado, ya que es un enlace que se realiza después de la compilación, este enlace permite enviar un mensaje a un objeto, y este último decide, durante la ejecución, como responder al mismo.

7. **Persistencia:** es la propiedad que tienen los objetos de poder quedarse vigentes en el tiempo y en el espacio, es indispensable en la mayoría de las aplicaciones (bases de datos orientadas a objetos).

1.5.2 Relaciones entre objetos

Relaciones de uso	Las relaciones de uso expresan la forma en que unos objetos utilizan a otros, las formas son las siguientes: A) Actor: un objeto que puede operar sobre otros objetos. B) Servidor: es un objeto que puede ser operado por otros objetos; pero el nunca puede operar sobre los demás. C) Agente: es un objeto que puede operar sobre los demás objetos y también puede ser operado por otros.
Relaciones de contenido	Se crea esta relación cuando un objeto está compuesto por otros objetos. Existe cierta interdependencia entre ambas relaciones entre objetos. A veces es conveniente tener objetos compuestos por otros, porque esto reduce el número de objetos en un sistema, pero a la vez causa estrecha dependencia entre objetos que dificultan otros usos.

Figura 1.12 Relaciones entre los objetos

1.6 CLASES

Muchos objetos del mundo tienen características y operaciones que ejecutan razonablemente similares.

La realización software de objetos del mundo real están categorizadas de la misma forma. Todos los objetos son miembros de una clase más amplia y heredan la estructura de datos privados y las operaciones que se han definido para esa clase. Por lo tanto una clase es:

Un conjunto de objetos que tienen las mismas características.

Los objetos (folletos, informes, enciclopedia) pueden definirse como instancias de libros (ver figura 1.13). El uso de clases y herencia es esencialmente importante en la moderna ingeniería de software.

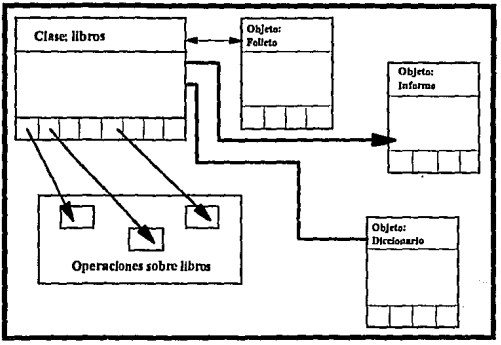


Figura 1.13 Objetos, clases y herencia

1.6.1 Características de una clase

Las clases poseen ciertas características que permiten entender los aspectos de las diferentes fases del ciclo de vida de un sistema desde el punto de vista de orientación a objetos (ver figura 1.14).

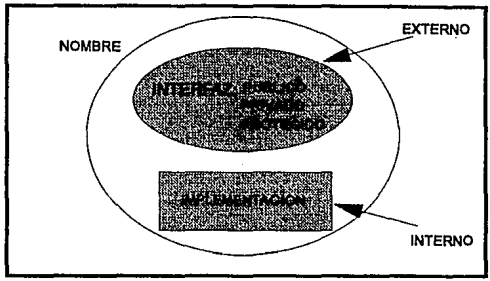


Figura 1.14 Una Clase

- Las clases permiten diseñar los sistemas de manera modular y jerárquica, encapsulando datos y operaciones en objetos.
- La clase se interpreta también como una abstracción del concepto de objeto. A un objeto, el cual pertenece a una clase, se le denomina también instancia de esa clase.

En otros términos, una *clase* es la declaración de las propiedades (estructuras de *datos* + *operaciones*) que tendrán los objetos pertenecientes a la clase. El concepto de clase permite abstraer las características comunes de los objetos que la componen.

1.6.2 Elementos que conforman una clase

- **INTERFAZ:** parte de una clase en la cual se describen los servicios de la clase y su forma de interacción con otras clases, entendiendo por servicios las operaciones o datos que puedan ser usados por los clientes de la clase.

  La interfaz de una clase proporciona su vista externa. Generalmente consta de los encabezados de las operaciones aplicables a instancias de esta clase, pero en algunas ocasiones puede incluir otro tipo de información, por ejemplo: otras clases, constantes, variables etc.
  Dependiendo de la información que proporcionan, se pueden distinguir a dos tipos de interfaces:
  a) Interfaz Sintáctica: declara el nombre de los servicios de una clase. En el caso de las operaciones sus parámetros. La declaración es meramente sintáctica y no precisa el significado semántico de los servicios.

b) Interfaz Semántica: aparte de la información proporcionada por la interfaz sintáctica, incluye la especificación del significado semántico de los servicios de una clase. Se utilizan notaciones formales para este fin, generalmente en forma de predicados.

La interfaz permite hacer de una clase una entidad independiente con límites bien definidos y es además la que proporciona la especificación de los servicios que la clase ofrece al exterior. La interfaz habilita la compilación por separado facilitando el trabajo en equipo al desarrollar un sistema.

- **IMPLEMENTACION:** comprende el código de los servicios descritos en la interfaz.
  La implantación de una clase es su vista interna. En principio, contiene las implantaciones de las operaciones definidas en la interfaz.
  La parte de implantación de una clase puede relacionarse con su interfaz de diversas formas. En general, las relaciones entre interfaces y las implementaciones que se encuentran acopladas, formando sólo una entidad, y las que se encuentran desacopladas como dos entidades independientes una a otra. Las interfaces desacopladas facilitan la creación de bibliotecas de clases reutilizables y el desarrollo independiente de sistemas, ya que una clase que depende de otra requiere únicamente la información de la interfaz para su compilación.

1.6.3 Tipos de clases

Superclase	Es una clase de la cual se hereda el comportamiento específico
Subclase	Es una clase que hereda el comportamiento de otra clase. Por lo general aumenta o re define la estructura y el comportamiento de sus superclases.
Clase Base	Es una clase que no tiene superclases.
Clase Abstracta	Es una clase cuyo comportamiento no está totalmente definido y por tanto no tiene sentido generar objetos de esta clase.

Figura 1.15 Diferentes tipos de clases

1.6.4 Relaciones entre clases

Las clases no existen aisladas, están relacionadas formando una estructura del diseño de un sistema, las relaciones que se encuentran son las siguientes.

Generalización : Relación *tipo_de*

Agregación : Relación *parte_de*

Asociación : Conexión semántica

Las anteriores relaciones se manifiestan de diferente forma en los lenguajes de programación:

Herencia (Generalización + Asociación)

Instanciación (Generalización + Asociación)

Metaclase (Generalización de clases)

Relación de herencia	La herencia es una relación entre clases en la que una clase comparte la estructura y/o comportamiento definidos en una o más clases obteniéndose la ***herencia sencilla*** o ***múltiple***.(fig. 1.17 y 1.18).
Relación de uso	Las dos principales relaciones de uso son: a) La interfaz de una clase usa a otra clase. b) La implementación de una clase usa a otra clase.
Relación de instanciación	Es una relación entre clases donde el comportamiento de una clase puede ser parámetrizado por otra clase. Ejemplos: Listas, colas, pilas, árboles, etc.
Relación de metaclases	Una metaclase es una clase cuyos objetos son clases. Esto permite una manipulación uniforme de todos los objetos en un sistema.
Relación entre clases y objetos	Las clases en la mayoría de los lenguajes son estáticas, forman parte del texto del programa y no cambia durante su ejecución. Mientras que los objetos son totalmente dinámicos, se crean, cambian su estado y se destruyen dinámicamente.

Figura 1.16 Tipos de relaciones entre clases

Herencia Múltiple: es la propiedad de una clase para heredar propiedades de más de una clase base. Muchos lenguajes de programación orientados a objetos no soportan herencia múltiple.

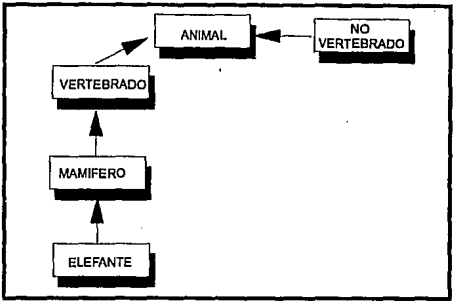


Figura 1.17 Herencia sencilla

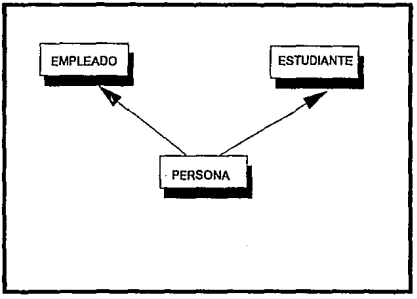
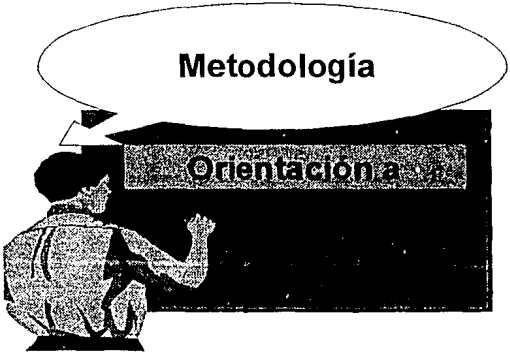
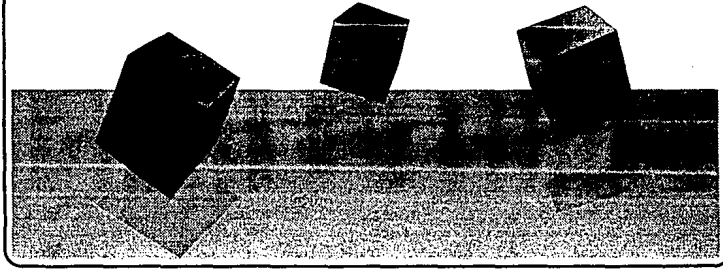


Figura 1.18 Herencia múltiple

CAPITULO II

Método de Grady Booch





METODO DE GRADY BOOCH

Considerando un mundo de software en pleno desarrollo y creatividad creciente, todos los autores interesados en este concepto han ido estableciendo nuevos principios que enseñan un panorama diferente de software.

Uno de los autores que ha tenido principalmente este interés por el software, es Grady Booch que con su Método de Grady Booch para la generación de software Orientado a Objetos, es uno de los precursores de la llamada **"Orientación a Objetos"**. Por ello es que en este capítulo se toma como referencia su punto de vista en el tema, para considerar su propia metodología, además de tomar en cuenta la metodología clásica como antecedente, la cual ha sido utilizada hasta el momento.

Para este caso el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requerimientos de todos los elementos del sistema y luego asignando algún subconjunto de estos requerimientos al software. Esta visión del sistema es esencial cuando el software debe interrelacionarse con otros elementos tales como hardware, personas y bases de datos. Por esta razón es necesario descubrir las fases de un ciclo de vida de un software desde el punto de vista de orientación a objetos y observar las diferencias que pueden llegar a existir entre lo ya establecido y la orientación a objetos, ya sean favorables o desfavorables para la generación de software.

En este capítulo se explicaran las fases correspondientes a los dos ciclos de vida para la generación de software, es decir, desde el punto de vista clásico o desde el punto de vista orientado a objetos; con el objeto de describir de manera resumida y general la metodología de orientación a objetos utilizada por Grady Booch.

2.1 METODO CLASICO

En la actualidad existe un método clásico para la realización de software, el cual consiste en el cumplimiento de las fases que se muestran en la figura 2.1. El modelo de fases divide el ciclo de vida del producto de programación en una serie de actividades sucesivas; cada fase requiere información de entrada, métodos y resultados, todos ellos bien definidos. Se necesitan recursos para terminar los métodos de cada fase, y cada una de ellas se efectúa mediante la aplicación de los métodos explícitos, herramientas y técnicas.

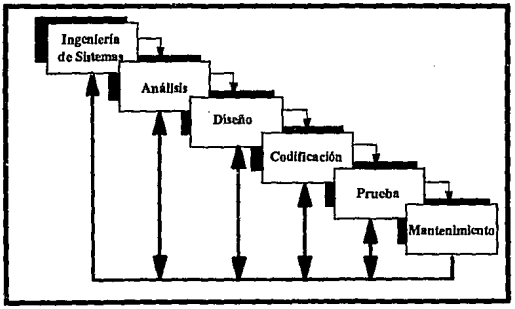


Figura 2.1 Modelo de fases para el desarrollo de software (clásico)

2.1.1 Análisis

Cada método de análisis tiene una única notación y punto de vista. Sin embargo, todos los métodos de análisis están relacionados por un conjunto de principios fundamentales:

1. El dominio de la información, así como el dominio funcional de un problema debe ser representado y comprendido.

2. El problema debe de subdividirse de forma que se descubran los detalles de una manera progresiva (o jerárquica).

3. Deben desarrollarse las representaciones lógicas y físicas del sistema.

En general puede decirse que esta fase consiste en efectuar un análisis completo del problema o sistema existente, con la finalidad de proponer un modelo adecuado para su mejor solución.

2.1.2 Diseño

El diseño de software es realmente un método multipaso que se enfoca sobre tres atributos distintos del programa: estructura de datos, arquitectura de software y detalle procedimental. El método de diseño traduce los requerimientos en una representación de software que pueda ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación. El diseño se documenta y forma parte de la configuración del software.
Con el fin de permitir la implantación y el uso de un sistema por parte de cierto número de personas, es necesario contar con una serie de documentación que describa tanto los aspectos estáticos, la *estructura de la base de datos*, como los aspectos de procedimientos, que realizan las transformaciones a los datos.

Para el aspecto estático es adecuada una imagen de los datos en términos de archivos, registros, campos y de la relación entre los datos contenidos en estos elementos.

Dicha definición de la estructura de datos puede adoptar la forma de un documento que sea utilizado por las personas que programan las operaciones de archivo de esta base de datos. Es aún mejor materializar las definiciones de estructuras en forma de un conjunto de códigos legibles para la computadora; esto es, un esquema que guíe automáticamente los métodos de archivo.

La descripción de los procedimientos puede darse como una fórmula, una descripción de secciones de programas a ejecutar, o un diagrama de flujo.

2.1.3 Codificación

El diseño debe traducirse en una forma legible para la máquina. El paso de la codificación ejecuta esta tarea. Si el diseño se ejecuta de una manera detallada, la codificación puede realizarse mecánicamente.

2.1.4 Prueba

Una vez que se ha generado el código, comienza la prueba del programa. La prueba se enfoca sobre la lógica interna del software, asegurando que todas las sentencias se han probado, y sobre las funciones externas, esto es, realizando pruebas para asegurar que la entrada definida producirá los resultados que realmente se requieren.

2.1.5 Mantenimiento

El software sufrirá indudablemente cambios después de que se entregue al cliente. Los cambios ocurrirán debido a que se han encontrado errores, a que el software debe adaptarse por cambios del entorno externo, a que el cliente requiere aumentos funcionales o del rendimiento. El mantenimiento del software se aplica a cada uno de los pasos precedentes del ciclo de vida un programa existente en vez de a uno nuevo.

Todas las fases que constituyen este método clásico son de suma importancia que se lleven acabo de acuerdo al lugar que ocupan; y aunque cada una de ellas tiene muy bien definida su actividad y su frontera con respecto a la siguiente, todas estas fases son probablemente hasta cierto punto dependientes unas de otras, ya que ninguna se puede realizar aislada sin tomar en cuenta a las otras como punto de referencia.

Actualmente este ciclo de vida clásico para la generación de software es el más empleado debido a la escasez de otras metodologías que sean realmente confiables y específicas. Por ello que en los últimos años se han preocupado varios investigadores de software por la definición de otras

perspectivas tanto de metodologías, métodos de programación y lenguajes. Considerando a uno de ellos, el más interesado en este punto de la orientación a objetos, siendo Grady Booch.

2.2 METODO ORIENTADO A OBJETOS

Por lo que respeta al método orientado a objetos en el ciclo de vida para la generación de software; el análisis y el diseño son las primeras fases en la producción de un sistema de software en las cuales se trata de definir con precisión cuál es el problema y la mejor manera de solucionarlo.

El análisis y diseño son las primeras etapas en el desarrollo de un sistema y su éxito se basa esencialmente en estas dos fases.

En particular, el análisis, como su nombre lo indica, significa analizar el problema del mundo real y crear su modelo abstracto, el cual se conoce como la representación del dominio del problema. La fase de diseño busca la definición de la solución. Por lo general, esta fase utiliza la representación abstracta del problema y se extiende para lograr la representación de la solución, misma que se conoce como la representación del dominio de la solución (figura 2.2) que incluye, por lo general, la representación del dominio del problema sumándole los conceptos indispensables para lograr la solución. Por ejemplo, la descripción de la interfaz con el usuario del sistema forma parte del dominio de la solución pero no del problema mismo.

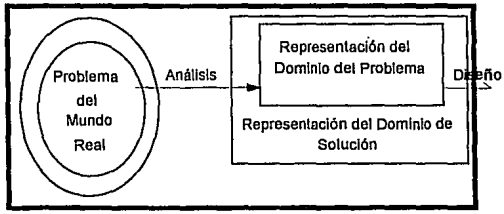


Figura 2.2 Pasos correspondientes al análisis y diseño de sistemas

Si a los métodos de análisis y diseño les añadimos el término orientado a objetos, lo que se esta diciendo es que las representaciones de los dominios del problema y de la solución están expresadas en los términos del modelo de objetos (mencionado en el capítulo I).

La fase del análisis orientada a objetos es la que se encarga de descubrir los objetos semánticos dentro del dominio del problema, esto se representa en la figura 2.3, es decir, las abstracciones que representan los conceptos que tienen un significado claro en la descripción del problema, como, por ejemplo, cliente, cajero, producto, servicio, etc.

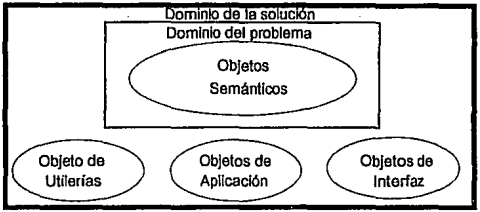


Figura 2.3 Diversos objetos en el dominio de la solución

En la figura 2.4 se observan las fases de desarrollo de un sistema utilizando el método orientado a objetos.

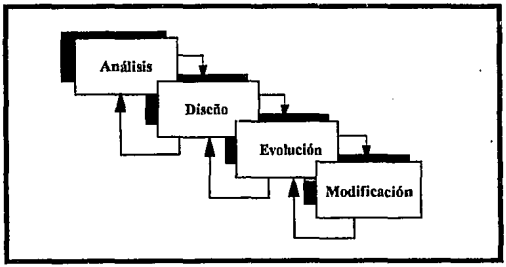


Figura 2.4 Modelo de fases orientado a objetos

Grady Booch considera que a diferencia del método clásico de desarrollo de software que consta de cinco fases, en este método que él plantea sólo considera cuatro fases considerando como las principales el análisis y diseño. Este autor define estas fases como sigue:

2.2.1 Análisis

Une a los usuarios y a los diseñadores. Debe proporcionar una descripción completa del problema, legible y revisable por las partes interesadas y verificable contra la realidad.

Las técnicas de ingeniería de software orientadas a objetos han generado un amplio interés en los últimos años. Los enfoques orientados a objetos para la definición y partición de un problema son prácticamente equivalentes en su aplicación al análisis de requerimientos de los problemas. De hecho la definición de objetos y operaciones es una forma excelente para comenzar el análisis de los dominios funcionales y de información. Además una visión orientada a objetos, conduce por sí mismo de una forma agradable, al principio de partición.

Un objeto puede verse como un elemento de información y una operación, tal como un método o función que se aplica a uno o más objetos.

El método de análisis orientado a objetos es una técnica general de definición de los objetos semánticos del dominio del problema.

El método de análisis orientado a objetos esta conformado por las siguientes actividades:

1. El software asignado (o el sistema entero) se describe usando una estrategia informal. La estrategia no es más que una descripción en lenguaje natural de la solución del problema que hay que resolver, mediante el software representado a un nivel consistente de detalle. La estrategia informal puede ser establecida en forma de párrafos sencillos, gramaticalmente correctos.

2. Los objetos se determinan subrayando cada nombre o cláusula nominal e introduciéndolo en una tabla sencilla. Deben anotarse los sinónimos. Si se requiere que los objetos se implementen como una solución, entonces es parte del espacio de solución; en otros casos, si un objeto es necesario sólo para describir una solución, es parte del espacio del problema.

3. Los atributos de los objetos se identifican subrayando todos los adjetivos y luego asociándolos con sus objetos respectivos (nombres).

4. Las operaciones se determinan subrayando todos los verbos, frases verbales y predicados (una frase verbal índica un est condicional) y relacionando cada operación con el objeto apropiado.

5. Los atributos de las operaciones se identifican subrayando todos los adverbios y luego asociándolos con sus operaciones respectivas (verbos).

Estas actividades no tienen porque realizarse secuencialmente, es decir, después de identificar algunas clases y atributos en el dominio del problema podemos pasar a la fase de colocación para ordenar, en cierta forma, los conceptos identificados, y luego se puede regresar nuevamente a la fase de identificación para buscar nuevos candidatos para los objetos semánticos. Normalmente, el método de análisis es iterativo, donde las actividades de identificación, colocación y especificación del comportamiento se repiten e intercalan de manera aleatoria.

Ahora bien concretando lo anterior se puede resumir el análisis de acuerdo a estos tres puntos:

1. **Identificación de las clases de objetos semánticos, los atributos y las operaciones que describen el comportamiento de los objetos.**

2. **Colocación de los atributos y las operaciones dentro de las clases, así como la definición de las relaciones de generalización, agregación y asociación entre clases.**

3. **Especificación del comportamiento dinámico de los objetos.**

Durante el análisis orientado a objetos las clases y objetos se derivan usualmente de los conceptos representados en la figura 2.5.

Cosas tangibles	(autos, sensores, ...)
Papeles que se representan	(madre, profesor, ...)
Eventos que ocurren	(aterrizaje, petición, ...)
Interacción	(préstamo, encuentro, ...)
Organizaciones	(escuela, ONU, ...)

Figura 2.5 Conceptos

Se sugiere la siguiente convención (figura 2.6) para asignar nombre a los objetos y a las clases.

Objeto	sustantivo (El Sensor, Un Sólido, ...)
Clase	nombre común (Sensores, Sólidos, ...)
Operaciones	verbos activos (Dibujar, Mover, ...)
Operaciones de selección	preguntas (Está abierto,...)

Figura 2.6 Nombre de las abstracciones

Las abstracciones sirven para crear el modelo, pero esto no es suficiente, hay que añadirle el comportamiento observable del sistema

Se llama mecanismo a cualquier estructura mediante la cual interactúan los objetos para satisfacer los requisitos de un problema, un ejemplo de mecanismos se observa en la figura 2.7.

Los mecanismos determinan de que manera interactúan las instancias de las clases.
La definición de los mecanismos es una decisión estratégica en el diseño de un sistema.

MS-DOS	monolítico
UNIX	núcleo y capa (shell)
THE	jerarquía de métodos por capas

Figura 2.7 Sistemas operativos con diferentes mecanismos de interacción

2.2.2 Diseño

Puede comenzar aún cuando la fase del análisis no esté concluida. Se sugiere analizar un poco y luego diseñar un poco. Debe concluir tras haber establecido las abstracciones claves y mecanismos importantes, dejando para la implantación aquellos aspectos de diseño que tienen poca o ninguna relación con la conducta observable del sistema; cuando las abstracciones clave son tan simples que no requieren mayor descomposición y pueden elaborarse con los componentes reutilizables existentes.

El diseño orientado a objetos como otras metodologías de diseño orientadas a la información, crea una representación del dominio del problema en el mundo real y lo transforma en un dominio de solución que es software. A diferencia de otros métodos, el diseño orientado a objetos da como resultado un diseño que interconexiona los objetos de datos (elementos de datos) y las operaciones de procesamiento de forma que modulariza la información y el procesamiento en vez de sólo el procesamiento.

La naturaleza única del diseño orientado a objetos está ligada a su habilidad para construir, basándose en tres conceptos importantes de diseño del software, **abstracción, ocultación de la información y modularidad**. Todos los métodos de diseño buscan la creación de software que exhiba estas características fundamentales, pero sólo el diseño orientado a objetos da un mecanismo que facilita al diseñador adquirir los tres sin complejidad o compromiso. Ahora bien se consideran también criterios como él de:

Autor: Wiener y Sincovec

Ya no es necesario para el diseñador de sistemas convertir el dominio del problema en estructuras de datos y control predefinidas, presentes en el lenguaje de implementación. En vez de ello, el diseñador puede crear sus propios tipos abstractos de datos y abstracciones funcionales y transformar el dominio del mundo real en estas abstracciones creadas por el programador. Esta transformación incidentalmente; puede ser mucho más natural debido al virtualmente ilimitado rango de tipos abstractos que pueden ser inventados por el diseñador. Además el diseño de software se separa de los detalles de representación de los objetos de datos usados en el sistema. Estos detalles de representación pueden cambiarse muchas veces, sin que se produzcan efectos inducidos en el sistema de software global.[7]

Es prematuro decir que el diseño orientado a objetos es una metodología compresiva. Sin embargo, los conceptos involucrados en el diseño orientado a objetos representan un importante y único punto de vista del diseño de software.

El diseño orientado a objetos crea un modelo del mundo real que puede ser realizado en software. Los objetos suministran un mecanismo para representar el dominio de la información, mientras que las operaciones describen el procesamiento asociado con el dominio de la información. Los mensajes (un mecanismo de interfaces) dan la forma en la que se llama a las operaciones. La característica distintiva del diseño orientado a objetos es que los objetos "conocen" qué operaciones pueden aplicarse a ellos. Este conocimiento se adquiere combinando las abstracciones de datos y procedimientos en una única componente de programa llamada a objeto o paquete.

El diseño orientado a objetos ha evolucionado como resultado de una nueva clase de lenguajes de programación orientados a objetos. Consecuentemente, las representaciones del diseño orientado a objetos son más propensas que otras a una dependencia del lenguaje de programación.

La metodología del diseño orientado a objetos consta de un método de tres pasos: requiere que el diseñador establezca el problema, defina una estrategia informal de solución y formalice la estrategia identificando los objetos y operaciones, especificando las interfaces y dando los detalles de implantación para las abstracciones de datos y procedimientos. Los tres pasos del diseño orientado a objetos se aplican recursivamente hasta que se crea un diseño detallado o hasta que se implementa el software como código. Los diseños se representan usando un lenguaje de diseño de programa y una de las distintas notaciones gráficas.

---

[7] Roger, S. Pressman, "Ingeniería de Software un Enfoque Práctico", Segunda Edición, Enero 1990.

Según Booch en la fase actual de evolución de la orientación a objetos, la metodología diseño orientado a objetos combina elementos de las tres categorías de diseño; diseño de datos, diseño arquitectural y diseño procedural. Para la identificación de los objetos, se crean abstracciones de datos. Definiendo operaciones, se especifican los módulos y se establece una estructura para el software. Desarrollando un mecanismo para usar los objetos.

Booch propone los siguientes pasos para el Diseño Orientado a Objetos:

1. **Definir el problema.**

2. **Desarrollar una estrategia informal para la realización software del dominio del problema en el mundo real.**

3. **Formalizar la estrategia usando los siguientes subpasos:**

a)Identificar los objetos y sus atributos.
b)Identificar las operaciones que pueden aplicarse a los objetos.
c)Establecer interfaces para mostrar las relaciones entre los objetos y las operaciones.
d)Decidir los aspectos de diseño detallado que harán una descripción de la implementación para los objetos.

4. **Repetir los pasos 2, 3 y 4 recursivamente hasta que se cree un diseño completo.**

Tomando en cuenta las características de las dos primeras fases del método de Grady Booch podemos decir:

> ***En el análisis orientado a objetos se hace un modelo en base a la identificación de clases y objetos que forman el vocabulario del dominio del problema.***
> ***En el diseño orientado a objetos se inventan las abstracciones y mecanismos que proporcionan el comportamiento que requiere el modelo.***
> ***De hecho el análisis orientado a objetos está considerado como el frente ideal para el diseño orientado a objetos. La frontera entre ambos es difusa.***

2.2.3 Evolución

Comprende los aspectos clásicos de programación, verificación e integración. El método de desarrollo consiste en una producción incremental de series de prototipos, que evolucionan a la implantación final. Los posibles cambios de diseño son:

- **Añadir una clase**
- **Cambiar la implantación de una clase**
- **Cambiar la representación de una clase**
- **Reorganizar la estructura de clases**
- **Cambiar la interfaz de una clase**

2.2.4 Modificación

Un programa que se usa en un ambiente real, necesariamente debe de cambiar. Los cambios que se le practican difieren un poco de los requeridos en la evolución del sistema, pues contemplan por lo general la introducción de nuevas funcionalidades que no estaban previstas en el problema original.

El método de diseño orientado a objetos, cuyo objetivo es hacer modelado del dominio, incluye una fase de optimización.

La optimización de clases, puede consistir en la reestructuración de las jerarquías y relaciones entre las clases debido al descubrimiento de ciertas posibilidades de mejorar el diseño, como son, por ejemplo, la reutilización de clases ya existentes, la generalización introduciendo clases abstractas, la partición de las clases demasiado grandes, etc.

2.3 REPRESENTACION EN EL ANALISIS Y DISEÑO ORIENTADO A OBJETOS

Las fases de análisis y diseño para la generación de software orientado a objetos deben proporcionar cierta notación para representar la información captada en sus distintas fases. Las notaciones varían mucho, van desde las representaciones gráficas de diversa índole hasta los formularios, diccionario o simple texto. Lo que cabe señalar es que, por lo general, se necesitan dos

tipos de notación para representar dos aspectos importantes de un sistema. Estos aspectos son: su *estructura estática y su comportamiento dinámico*. A veces se le llama la *vista estática y la vista dinámica* de un sistema.

La notación para representar la vista estática de un sistema debe proporcionar herramientas para denotar, clases, objetos, sus atributos y su comportamiento, al igual que las relaciones de generalización, agregación y asociación que rigen entre los objetos y las clases.

La vista dinámica de un sistema debe expresar la comunicación entre objetos, el flujo de control, la temporización de los eventos, el cambio de estado de los objetos, etc. Para representar la vista dinámica, en muchas ocasiones se aprovechan notaciones ya existentes, como en el caso de los *diagramas de transición de estado* o los *diagramas de tiempo*.

2.3.1 Notación

La notación es un medio para documentar el análisis y diseño de un sistema. Es importante contar con una notación bien definida, expresiva, estándar e independiente de cualquier lenguaje de programación.

Es imposible usar un solo diagrama para especificar un sistema complejo. Se necesitan especificar varios aspectos del sistema por separado. Por un lado, se especifica la estructura estática (clases, objetos, módulos, métodos) y por el otro, la dinámica del sistema (transición de estados, tiempo). Las especificaciones estática y dinámica cuentan con los diagramas que se presentan en la figura 2.8.

ESPECIFICACION ESTATICA	Lógica	A)Diagrama de clases B)Diagrama de objetos
	Física	Diagrama de módulos Diagrama de métodos
ESPECIFICACION DINAMICA		C)Diagrama de transición de estado D)Diagrama de tiempo

Figura 2.8 Especificación estática y dinámica

A) Diagramas de clases

Se usa para mostrar la existencia de clases y las relaciones entre ellas. Las notaciones que se utilizan para representar el uso de las clases en el diseño y análisis orientado a objetos se encuentran en la figura 2.9.

ICONO	DESCRIPCION
Nombre	CATEGORIA DE CLASE
Nombre	CLASE
Nombre	UTILERIA DE CLASE
	USO EN INTERFAZ
	USO EN IMPLANTACION
	INSTANCIACION (TIPO COMPATIBLE)
	INSTANCIACION (TIPO NUEVO)
	HERENCIA (TIPO COMPATIBLE)
	HERENCIA (TIPO NUEVO)
	METACLASE
	NO DEFINIDA

2.9 Notaciones utilizadas en el análisis y diseño orientado a objetos

- **Relaciones y sus cardinalidades.**

Uso en interfaz 0: se indica mediante una línea doble y círculo del lado de la clase que usa en su interfaz elementos de la otra clase.

Uso en implantación 1: se indica mediante una línea doble y círculo oscuro del lado de la clase que usa en su implantación elementos de la otra clase.

Durante la fase del diseño generalmente es suficiente manejar solamente la primera relación y luego, cuando se toman las decisiones de la implantación, añadir la segunda.

Instanciación (tipo compatible): la relación de instanciación entre las clases se denota mediante una línea no continua con la flecha dirigida hacia la clase que es una instanciada.

Instanciación (tipo nuevo): es parecida a la anterior. Se usa para lenguajes con unidades de tipo genéricas.
Es mejor usar ambas relaciones de instanciación solamente para los lenguajes que tienen algún tipo de parametrización de clases o unidades genéricas.

Herencia (tipos compatibles): se denota mediante una sola línea con la flecha apuntado a la clase de la cual se hereda. Por lo general, se evita la especificación explícita de la herencia de la clase base.

Herencia (tipos nuevos): se denota casi igual que la anterior poniendo una rayita transversal al inicio de la línea. Este tipo de relación se usa solamente para tipos derivados de ADA.

Metaclase: para esta relación se usa en la notación original de Booch una línea en gris con flecha apuntando hacia ella.

No definida: esta relación entre clases, denotada por una línea no continua, conviene utilizarla en las primeras fases del diseño, cuando todavía no se está seguro de cuál es el tipo exacto de la relación. Mientras avanza el diseño, este tipo de relación queda sustituido por una de las relaciones antes mencionadas. Cada raya que denota una relación puede acompañarse con una etiqueta que explique su significado.

- **Utilerías de clases**

En los lenguajes que permiten la existencia de subprogramas fuera de clases (C++, ADA, etc.) es útil tener una representación para ellos, o sus conjuntos, en el diagrama de clases. Para distinguirlos de clases, su ícono se marca con una sombra como se muestra a continuación en la figura 2.10.

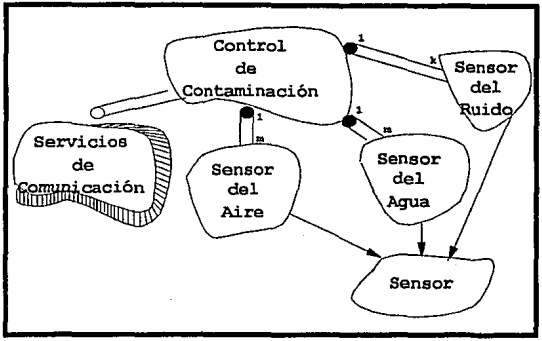


Figura 2.10 Diagrama de clases utilizando una utilería de clases

- **Categorías de clases**

En un sistema puede haber cientos de clases. Por lo tanto sería poco práctico incluirlas en un solo diagrama. Por lo consiguiente, se propone agrupar las clases en categorías y hacer diagramas de las misma. Un ejemplo de ello se muestra en la figura 2.11.

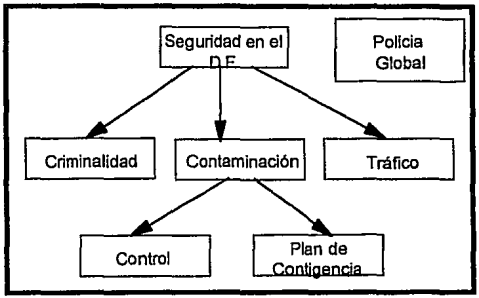


Figura 2.11 Ejemplo del diagrama de categorías de clases

Para cada categoría de clases son importantes sus reglas de visibilidad, es decir, los nombres que exporta, los nombres privados y los nombres importados de otras categorías. En el

diagrama de categorías se pueden indicar esas reglas enlistando todos estos nombres y enmarcando a los exportados así como subrayando a los importados, dejando sin ninguna marca a los privados.

La relación de importación es la única que importa a ese nivel de diseño y se puede expresar con una raya con flecha. También se necesita tener categorías de clases que sean visibles para las demás. En estos casos se sugiere anotar la palabra *global* en la esquina izquierda del ícono de la categoría.

- **Esquemas de clases**

Para complementar la información de los diagramas de clases se necesita una documentación más precisa, misma que se denomina esquema de clase. Un esquema de clase, en su versión general, contiene la información que muestra a continuación la figura 2.12.

Nombre	Identificador
Documentación:	texto
Visibilidad:	exportado/privado/importado
Cardinalidad:	0/1/n
Jerarquía:	
Superclases:	lista de nombres de clases
Metaclase:	nombre de clase
Parámetros genéricos:	lista de parámetros
Interfaz:	(Público/Protegido/Privado)
Usa:	lista de nombres de clases
Campos:	lista de declaraciones
Operaciones:	lista de declaraciones
Implantación:	
Usa:	lista de nombres de clases
Campos:	lista de declaraciones
Operaciones:	lista de declaraciones
Máquina de estados finitos:	diagrama de transición
Concurrencia:	secuencial/bloqueo/activo
Complejidad de espacio:	texto
Persistencia:	estática/dinámica

Figura 2.12 Esquema de clases

Por supuesto, no toda esa información tiene que aparecer en cada diseño. Algunas dependen del lenguaje de implantación particular y otras se concretizan mientras avanza el método de diseño.

- **Esquemas de utilerías de clases**

También se deben de especificar los subprogramas de utilerías. Sus esquemas son un poco más sencillos que las de las clases y se muestran a continuación en la figura 2.13.

Nombre	Identificador
Documentación:	texto
Visibilidad:	exportado/privado/importado
Parámetros genéricos	lista de parámetros
Interfaz:	
Usa:	lista de nombre de clases
Campos:	lista de declaraciones
Operaciones:	lista de declaraciones
Implantación:	
Usa:	lista de nombre de clases
Campos:	lista de declaraciones
Operaciones:	lista de declaraciones

Figura 2.13 Esquema de clases

- **Esquema de operaciones**

En la mayoría de los casos, en la fase avanzada del diseño se requiere de una definición más precisa de las operaciones que forman parte de las clases. Para este fin, se usa el esquema de las operaciones que se presenta a continuación en la figura 2.14.

Nombre	Identificador
Documentación:	texto
Categoría:	texto
Parámetros formales:	lista de declaraciones
Resultado:	nombre de la clase
Precondiciones:	texto
Acciones:	texto
Postcondiciones:	
Excepciones:	lista de declaraciones
Concurrencia:	secuencial/concurrente/múltiple
Complejidad de tiempo:	texto
Complejidad de espacio:	texto

Figura 2.14 Esquema de clases

B) Diagramas de objetos

Las clases son estáticas mientras que los objetos *dinámicos*. Con diagramas de objetos se reflejan principalmente las interacciones dinámicas entre objetos relacionadas, con el envío de mensajes en el diagrama de objetos deben de ser consistentes con el diagrama de clases.

El ícono que se usa para los objetos es el mismo que para las clases, sólo que representado con una línea continua y la relación de envío de mensaje se denota con una línea con etiqueta.

- **Esquemas de objetos y mensajes**

Al igual que con las clases, se completa el diseño de los objetos con una documentación con el siguiente formato, representado en la figura 2.15 para los objetos y la figura 2.16 para los mensajes.

Nombre	Identificador
Documentación	texto
Clase	nombre de la clase
Persistencia	persistencia/estático/dinámico

Figura 2.15 Esquema de objetos

Operación	Identificador
Documentación	texto
Frecuencia	no periódica/periódica
Sincronización	simple/síncrona/asíncrona

Figura 2.16 Esquema de mensajes

C) Diagramas de transición de estado

Usa el estado de una instancia 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. El estado inicial se representa con dos círculos concéntricos y el final con una línea gruesa. Los arcos del diagrama están etiquetados con nombres de las acciones o eventos.

D) Diagramas de tiempo

Se usan para mostrar las interacciones dinámicas entre varios objetos. Un diagrama de tiempo es una gráfica del tiempo contra los objetos involucrados en la interacción, como ejemplo a continuación se presenta la figura 2.17.

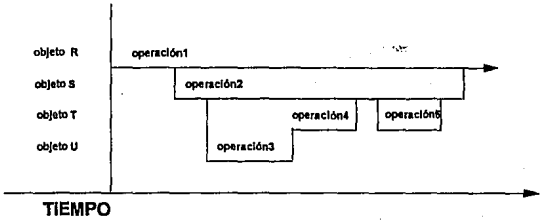


Figura 2.17 Ejemplo del diagrama de tiempo

NOTA:

Los diagramas de clases y de objetos sirven para documentar la estructura lógica de un sistema. Para complementar el diseño se necesita expresar la distribución física en módulos y métodos.

2.4 VENTAJAS Y DESVENTAJAS DEL METODO

Toda ideología innovadora debe de cumplir con ciertos requerimientos básicos y además ventajas sobre cualquier otro tipo de método. Sin embargo esto no significa que para este caso el método no tenga ningún riesgo. Por ello es fundamental realizar un análisis comparativo entre el método clásico para la generación de software y el método orientado a objetos para el mismo fin; y en particular para este método es necesario tomar en cuenta los siguientes aspectos:

ADMINISTRACION DE UN PROYECTO

En la asignación de recursos, comparada con el diseño clásico, el diseño orientado a objetos incurre en:

- **Gastos equivalentes en análisis y mayores en diseño.**
- **Gastos mucho menores en programación y verificación.**
- **Gastos considerablemente menores en integración (incremental).**
- **Recursos humanos equivalentes o menores.**
- **Un producto de mayor calidad.**

BENEFICIOS

- **Utiliza el poder expresivo de los lenguajes orientados a objetos.**
- **Favorece la reutilización de software.**
- **Los sistemas resultan flexibles al cambio.**
- **Reduce los riesgos de desarrollo.**
- **Es atractivo para la cognición humana.**

Se puede considerar también el punto de vista del siguiente autor:

Autor: Tom Lovet

El diseño orientado a objetos representa un método único para los ingenieros de programación.

Las raíces de los problemas del software pueden estar ligadas a la mayoría de los términos clásicos que describe nuestra industria -procesamiento de datos-. Se ha pensado que los datos y su procesamiento son dos "cosas" distintas, las cuales son algo fundamentales para nuestros negocios. Esta división puede ser más perjudicial de lo que se cree.

El diseño orientado a objetos presenta una forma de suprimir las ***"subdivisiones"*** entre datos y métodos. Al hacerlo, puede mejorarse la calidad de software.

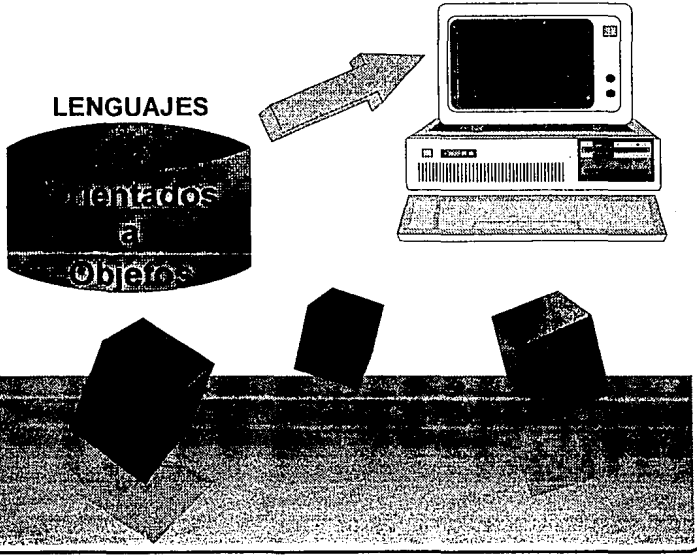
RIESGOS

- **Desempeño menor respecto a un sistema diseñado clásicamente.**
- **Costos iniciales de adopción de diseño orientado a objetos.**

Dado que es un método nuevo, requiere del uso de lenguajes de programación relativamente nuevos. Esto origina que los costos iniciales puedan llegar a ser un verdadero inconveniente para su adopción. Se sugiere empezar por aplicarlo en proyectos de bajo riesgo.

CAPITULO III

Programación y Lenguajes



PROGRAMACION Y LENGUAJES

La disciplina de la programación de computadoras ha sido por muchos años un lugar de oportunidad para el desarrollo del intelecto humano. Para llevar acabo dicha tarea existen diversos métodos de programación que fueron cambiando debido a las necesidades de los programadores, por desarrollar de alguna forma más rápida y eficiente hasta hoy en la actualidad, software reutilizable.

Gran parte del software que se viene realizando desde los años 50's requiere a parte de un ***método de programación*** (pasos necesarios para el desarrollo de un programa) un ***lenguaje*** (medio de comunicación entre la computadora y el programador), los cuales están basados en una serie de premisas o postulados que definen las características y licitantes de los mismos.

Antes de definir cuales fueron los inicios de la programación definiremos el concepto de programar. El ***"programar"*** es un proceso mental y complejo, dividido en varias etapas. La finalidad de la programación, así entendida, es comprender con claridad el problema que va a resolverse o simularse por medio de la computadora, y entender con detalle cuál va a ser el procedimiento mediante el cual la máquina llegará a la solución deseada.

La actividad de la programación es ante todo conceptual, y su campo de acción está centrado en tratar de definir, cada vez con mayor precisión, acercamientos que resuelvan el problema de manera virtual, es decir, que efectúen una especie de experimentos mentales sobre el problema a resolver o simular. El resultado de tales experimentos constituye una descripción de los pasos necesarios para encontrar la solución. Esta descripción, como cualquier otra, esta expresada en un lenguaje determinado.

En este capítulo se describen brevemente cuales han sido los métodos de programación, lenguajes y características desde el inicio de la computación hasta nuestros días.

3.1 METODOS DE PROGRAMACION

La programación vista desde los inicios de la computación fue el desarrollo de la misma mediante el uso del lenguaje de MAQUINA (0's y 1's).

A finales de los 50's y principios de la 60's surgió un cambio en el lenguaje utilizado para programar, pues el lenguaje de MAQUINA fue sustituido por el lenguaje ENSAMBLADOR y posteriormente por los lenguajes de alto nivel.

Resulta necesario mencionar que para la instrumentación de los postulados de un tipo de programación se requiere contar con una infraestructura básica en el hardware que permita la ejecución adecuada (rápida, eficiente y óptima) del mismo. Específicamente las herramientas que permiten la instrumentación de los diversos tipos de programación son el uso extensivo de la fase de interrupción de un CPU (Unidad Central de Procesamiento), el manejo dinámico de la memoria y la existencia de los ambientes de programación residentes.

Un punto muy importante es que aunque normalmente se asocia un tipo de programación a una sintaxis específica, inclusive a un lenguaje, en particular la forma de pensar es en si ajena a la sintaxis siendo más bien una cuestión de la semántica y de la forma de conceptualizar un problema. Esta es precisamente la razón por la cual en algunos lenguajes procedurales es posible programar usando un tipo de programación distinto *(este es el caso de PASCAL con objetos o C con ENSAMBLADOR).*

Actualmente existen diferentes formas de programar que generalmente son llamados métodos de programación.

3.1.1 Programación modular

Un programa no es más que una serie de instrucciones que le dicen a la computadora que lleve a cabo acciones específicas. Los programas pequeños puede construirlos un programador como una sola secuencia de instrucciones que efectúen la tarea deseada. Los programas de mayor tamaño no se pueden construir de esta forma, por lo que frecuentemente es necesario descomponer los programas grandes en componentes pequeños que puedan construirse independientemente y después juntarlos para formar el sistema completo. Esta forma de trabajo que vienen realizando la

mayor parte de personas que desarrollan software en los últimos 40 años, se denomina **"Programación Modular".**

El soporte más elemental para la programación modular se dio con la invención de la "Subrutina" a principios de los 50's. Una subrutina se crea al sacar una secuencia de instrucciones del programa principal y darle un nombre separado.

Las subrutinas proporcionan el mecanismo básico para la programación modular, pero además se requiere mucha disciplina para crear software bien estructurado, sino se cuenta con dicha disciplina es muy fácil escribir programas complicados que se resistan al cambio, difíciles de entender e imposibles de mantener. Es lo que con frecuencia pasó en los primeros años de la industria.

La programación modular es un método de diseño y tiende a dividir el problema total en partes perfectamente diferenciadas que pueden ser analizadas, programadas y puestas a punto por separado. La programación modular se basa en un diseño descendente (Top-down) como se muestra en la figura 3.1.

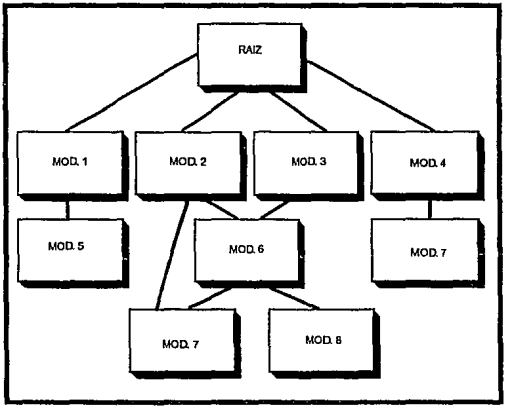


Figura 3.1 Diagrama ejemplificando la programación modular

El montaje de la red se hace en modo ascendente (Bottom-up). De este modo, un programador puede estar escribiendo por así llamarlo el módulo 8 mientras otro hace lo propio con

el módulo 7, una vez que ambos han terminado su trabajo, puede pensarse en escribir un módulo 6 ficticio que sólo llama a los módulos para comprobar su funcionamiento etc.

En la elaboración de un programa con métodos modulares aparecen las siguientes fases: ***Diseño descendente*** (Top_down) de la red de módulos, ***análisis de cada relación*** (parámetros que se envían a cada módulo y parámetros que éste devuelve), ***diseño de cada módulo*** y ***montaje ascendente*** (Bottom-up).

Disminuir complejidad	Se comprende que al dividir un problema en partes, se disminuye la complejidad de cada una de las unidades en forma más que proporcional.
Disminuir el costo	El costo de la programación disminuye automáticamente como consecuencia de la disminución de la complejidad y también por la metodología implícita.
Aumentar la claridad	Consecuencia de la disminución de la complejidad de cada unidad, se comprende que cada una de ellas ganará en claridad, resultado más fácil de entender para el propio programador y/o para terceras personas.
Aumentar la fiabilidad	En efecto, la disminución de complejidad y la metodología asociada, necesariamente comparten un aumento de la fiabilidad de los programas.
Aumentar el control del proyecto informático	Al resultar módulos acotados, cada uno de ellos puede analizarse, programarse y ponerse a punto por separado, minimizándose los problemas de puesta a punto final.
Facilitar la extendilidad	La ampliación de un programa construido con técnicas modulares es mucho más simple puesto que se limita a añadir módulos nuevos o ampliar solamente uno de los ya existentes, quedando el resto inalterado.
Facilitar las modificaciones y correcciones	Al quedar automáticamente localizadas en un módulo concreto.

Obtener recursos generalizados	La programación modular facilita la elaboración de procedimientos generales que podrán utilizarse en otras aplicaciones posteriores, cuyo programa principal no tendrá más que llamar al módulo correspondiente.

Figura 3.2 Objetivos de la programación modular

3.1.2 Programación estructurada

Los fundamentos del diseño procedimental se formaron a principios de los años 60's y se solidificaron con el trabajo de Edgar Dijkstra.

A finales de los 60's, el pobre estado del software disparo un esfuerzo concertado entre los científicos de la computación para desarrollar un estilo de programación consistente y disciplinado. *El resultado de ese esfuerzo fue el refinamiento de la programación modular en el enfoque conocido como "Programación Estructurada"*

Dijkstra y otros propusieron el uso de un conjunto de construcciones lógicas con las que podría formarse cualquier programa. Las construcciones reforzaban el "mantenimiento del dominio funcional". Esto es, cada construcción tenía una estructura lógica predecible, se entraba a ella por el principio y se salía por el final, y facilitaba al lector seguir el flujo procedimental. Las construcciones son: ***Secuencia*** implementa los pasos de procedimientos esenciales en la especificación de cualquier algoritmo, la ***Condición*** da la facilidad para seleccionar un procedimiento basado en alguna ocurrencia lógica y la ***Repetición*** suministra el bucle. Estas tres construcciones son fundamentales en la programación estructurada.

Las construcciones estructuradas se propusieron para limitar el diseño procedimental del software a un pequeño número de operaciones predecibles.

Las construcciones estructuradas son trozos lógicos que permiten a un lector reconocer los elementos procedimentales de un módulo en vez de leer el diseño o código línea a línea. La comprensión aumenta cuando se encuentran formas lógicas fácilmente reconocibles.

La programación estructurada corresponde a un método de programación, o forma de programar y se basa simplemente ***en limitar el conjunto de estructuras posibles de un ordinograma a unas pocas estructuras privilegiadas que, en general, tienden a estructurar el problema, procurando que su texto corresponda a su orden de ejecución.*** Los objetivos de este tipo de programación se muestran en la figura 3.3.

Disminuir complejidad	El modelo estructural del programa se expande en un modelo de diseño detallado que contiene las operaciones necesarias para resolver el problema.
Facilidad de legibilidad, prueba y mantenimiento	Se comprende mejor el problema al trabajar en módulos, y por lo tanto se utiliza el diseño descendente (Top_down) que incluye la habilidad de experimentar con prototipos sucesivos a medida que el sistema se desarrolla, es decir se van realizando las pruebas del sistema y la integración gradual de subsistemas.
Disminuir los costos	El costo de la programación disminuye automáticamente como consecuencia de la disminución de la complejidad y también por la metodología implícita.

Figura 3.3 Objetivos de la programación estructurada

La programación estructurada ha producido mejoras significativas en la calidad del software en los últimos 20 años, pero sus limitaciones son penosamente aparentes hoy en día. Uno de los problemas más serios es que rara vez es posible anticipar el diseño completo de un sistema antes de que sea implantado. Entre más grande sea el sistema, es más frecuente que se requiera reestructurarlo (ver figura 3.4).

Al atacar el análisis de un problema se pueden utilizar criterios de programación modular para dividirlo en partes independientes, utilizando métodos estructurados en la programación de cada módulo, probando por separado y realizando luego su montaje ascendente.

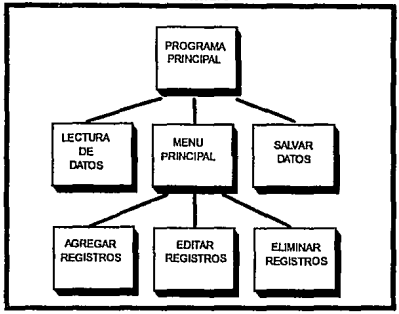


Figura 3.4 Diagrama ejemplificando la programación estructurada

3.1.3 Programación orientada a objetos

Este tipo de programación que se encuentra muy en boga hoy en día, surge en los años 60's con la definición del tipo CLASE en el lenguaje SIMULA 67, producto de los trabajos de los noruegos Dahl y Nygaard. Más tarde Alan Kay y Adele Goldberg en el trabajo con una interfaz amigable definen un nuevo lenguaje denominado SMALLTALK.

Lo importante de este tipo de programación está en la ruptura con la programación procedural que ha generado una concepción distinta de la programación, dejando de ser jerárquica (Top-down) y convirtiéndose en reticular (INDEPENDENT MODULES). Esta nueva visión plantea los siguientes principios:

- **El control debe ser distribuido.**
- **Los procedimientos deben estar desarrollados alrededor de metas definidas.**
- **Los procedimientos deben conocer lo mínimo unos de otros.**
- **Mensajes (avisos, sugerencias, quejas) deben poder ser enviadas entre los diversos elementos o procedimientos.**
- **Cada elemento debe ser lo suficientemente inteligente para encontrar sus propios errores.**

3.2 PROGRAMACION ORIENTADA A OBJETOS

La *metodología de programación* es la técnica que permite que la programación sea lo más eficaz posible en cuanto al desarrollo y mantenimiento.

La realización de un programa sin seguir un método de programación riguroso, aunque funcione, a la larga no será más que un conjunto más o menos grande de instrucciones. La definición de las diferentes etapas adolecen en general de indefinición y falta de continuidad (desarrollo y mantenimiento). Si se considera un programa sin tomar en cuenta un método predefinido podría caer en la siguiente relación de problemas y defectos:

- Los programas suelen ser excesivamente rígidos, presentado problemas para adaptarlos a las cada día más cambiantes configuraciones.
- Los programadores gastan la mayoría de su tiempo corrigiendo sus errores.
- Los programadores generalmente rehusan utilizar los programas y módulos ya escritos y en funcionamiento, prefiriendo escribir los suyos. La comunicación entre programadores es muy difícil.
- Un proyecto de varios programadores tiene normalmente varios conjuntos diferentes de objetivos.
- Un programador tiene sus propios programas convirtiéndose esta relación en algo inseparable.
- Las modificaciones en las aplicaciones y programas son muy difíciles de hacer, implican mucho tiempo y elevado costo de mantenimiento. Ello conduce a colocar "parches" que complican cada vez más el diseño inicial o bien a que el programa caiga en desuso y frente al elevado costo de actualización se opte por crear una nueva aplicación que sustituya a la existente.
- Deficiencias en la documentación.

En la actualidad, en la mayoría de los sistemas que se realizan, los problemas antes mencionados se encuentran con frecuencia. Debido a esto surge la necesidad de buscar nuevos horizontes que permiten de alguna forma encontrar una solución a dichos problemas, por ello hoy en día surge el concepto orientado a objetos *que pretende que los sistemas resulten flexibles al cambio, favorecer la reutilización del software y reducir los riesgos de desarrollo etc.*

3.2.1 Definición

El universo deja de ser poblado por procedimientos responsabilizándose por tareas sobre datos ajenos en favor de objetos. Donde la organización dinámica de procedimientos es jerárquica, la de objetos es más plana. Los objetos se comunican entre sí, por medio de mensajes, pidiendo unos a otros que efectúen tareas que a ellos les corresponden o que provean información sobre su estado actual. La unidad básica de operación sigue siendo el comando, y la ejecución de un programa, una secuencia de acciones. Las acciones son definidas como métodos de objetos que responden al paso de mensajes en vez de procedimientos invocados por llamados. La postura mental adoptada por un programador trabajando en la tradición de programación orientada a objetos prescinde del todo poderoso de la programación procedural.

Podemos definir a la programación orientada a objetos de la siguiente forma como:

Un nuevo método de diseño e implementación de sistemas de software, cuyo objetivo es el de aumentar la productividad del programador incrementando la extensibilidad y reutilización de software, además de controlar la complejidad y costo del mantenimiento del mismo.

De la misma forma como el término programación estructurada se refiere a una técnica de diseño y construcción de programas que enfatiza el uso de subrutinas y el manejo disciplinado de estructuras de control, el término programación orientada a objetos se refiere a una técnica de construcción de sistemas basados en la combinación de componentes, con límites bien establecidos, que enfatiza la creación de bibliotecas de componentes reutilizables, esta técnica cambia la perspectiva del desarrollo de sistemas.

La programación orientada a objetos induce al cambio en la cultura computacional, que va mucho más allá de la incorporación de nuevos mecanismos de abstracción en los lenguajes de programación. El verdadero impacto de la programación orientada a objetos está en el cambio de la perspectiva del desarrollo de sistemas, que pasa de ser una actividad basada en una boutique personalizada de subrutinas a ser una industria de componentes reutilizables.
Un componente es un módulo cuya interfaz con el exterior está claramente especificada. Un componente contiene una descripción puntual de la abstracción con la cual relaciona. La descripción enfatiza *que* es lo que hace el componente, presentado la información relevante y ocultando los detalles que son irrelevantes para el entendimiento de la abstracción.

En esta nueva perspectiva, el desarrollo de sistemas es una actividad que consiste en generar componentes que puedan ser reutilizados. Un sistema se construye en base a componentes previamente diseñados e implementados. Los componentes se seleccionan de una biblioteca y se especializan de acuerdo a requerimientos concretos. El objetivo final, a largo plazo, es amortizar el costo de desarrollo de sistemas mediante la reutilización de componentes. Esta perspectiva cambia, también, el ciclo de desarrollo de un sistema, así como las metodologías de análisis y diseño, la administración de sistemas y, en general, la cultura de desarrollo de sistemas.

3.2.2 Características

La programación orientada a objetos cuenta con conceptos básicos para la creación de la misma, estos conceptos se observan en la figura 3.5.

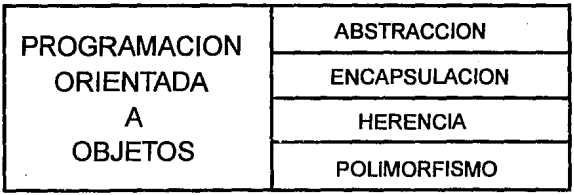
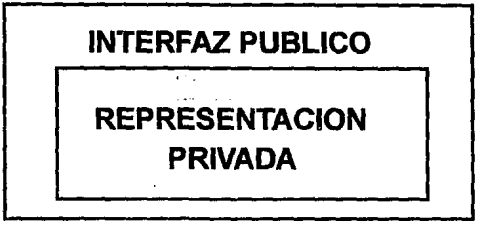


Figura 3.5 Características de la programación orientada a objetos

- **Abstracción de datos y objetos:** es el concepto de combinar la definición de tipos de datos y ocultación de datos. La idea de abstracción de datos se utiliza para crear un objeto que es una entidad que agrupa en él mismo, los datos y las operaciones que afectan a esos datos, o lo que es igual, las funciones que operan sobre esos datos.

- **Encapsulación:** el concepto de agrupar datos y funciones en un objeto se denomina encapsulación.

La encapsulación hace que el objeto tenga un interfaz público y una representación privada; este principio se conoce como *ocultación de la información.*



En la mayoría de los lenguajes se define una *clase* para describir el comportamiento del tipo abstracto de datos. En las clases pueden existir zonas *privadas*, de uso restringido a las funciones de la clase y zonas *públicas* de libre acceso fuera del ámbito de la clase. En terminología orientada a objetos, las operaciones que se definen para una clase se denominan *métodos*.

Un objeto es una variable declarada (instancia) de una clase específica. A los objetos se puede acceder sólo a través de su interfaz público, siempre que su acceso esté permitido. Se accede a un objeto enviándole un mensaje, que consta del nombre de una operación y los argumentos requeridos. Cuando un objeto recibe un mensaje, se realizan las operaciones solicitadas ejecutando un método (una función). Es decir, los datos no son accesibles desde el mundo exterior del objeto; el único medio para hacer algo a un objeto es llamando a una de las funciones que implementan el comportamiento del objeto.

Tipo de dato objeto	Clase
Objeto	Instancia de una clase
Funciones	Métodos

- **Herencia:** es la propiedad por la cual una clase (u objeto) heredan las propiedades de otra, o más correctamente una *clase* hereda el comportamiento de otra clase. La herencia impone una relación jerárquica padre-hijo entre clases, donde un hijo hereda de su padre. La clase padre se denomina con frecuencia clase *superclase* o clase *base*.

- **Polimorfismo:** es la propiedad por la cual dos o más clases de objetos responden al mismo mensaje cada uno de forma diferente. Esto significa que no necesita conocer quién le envía el mensaje necesita conocer cuántas clases diferentes de objetos han sido definidas para responder a ese mensaje particular.

El polimorfismo juega un papel importante en el paradigma de la programación orientada a objetos y facilita la resolución de problemas. Ayuda a simplificar la sintaxis de realización de la misma operación sobre una colección de objeto.

3.3 LENGUAJES

El lenguaje es un sistema de signos con que el hombre manifiesta lo que piensa o siente. Es el medio a través del cual el hombre se comunica, por medio de la palabra hablada o escrita, existiendo una diversidad de ellos. De esta forma también en el ámbito de la computación existe lo que se denomina **lenguaje de programación** y se puede definir de la siguiente forma: ***es el medio mediante el cual se puede uno comunicar con la computadora.*** Los lenguajes de programación son mecanismos notacionales que se usan para instrumentar productos de la programación.

3.3.1 Una visión de ingeniería sobre las características de los lenguajes

Como se ha dicho los lenguajes de programación son un vehículo de comunicación entre los humanos y las computadoras. El proceso de codificación -comunicación mediante un lenguaje de programación- es una actividad humana. Las características de ingeniería de un lenguaje tienen un impacto importante sobre el éxito de un proyecto de desarrollo de software.

Una visión de ingeniería del software sobre las características de los lenguajes de programación se centra en las necesidades que pueda tener un proyecto específico de desarrollo de software.

Las características desde el punto de vista de ingeniería son las siguientes:

- Facilidad de traducción del diseño al código.
- Eficiencia del compilador.
- Portabilidad del código fuente.
- Disponibilidad de herramientas de desarrollo.
- Facilidad de mantenimiento.

El paso de codificación comienza una vez que se ha definido, revisado y modificado el diseño detallado. En teoría, la generación de código fuente a partir de las especificaciones del diseño detallado debería ser algo directo. El grado de facilidad de la traducción del diseño al código proporciona una indicación de cómo se aproxima un lenguaje de programación a la representación del diseño, un lenguaje que implemente directamente las construcciones estructuradas, que incluya estructuras de datos sofisticadas, E/S especializada y manejo de cadenas, hará que la traducción del diseño al código fuente sea mucho más fácil.

Aunque los rápidos avances en velocidad del procesador y densidad de memoria han comenzado a disminuir la necesidad de "código super eficiente", muchas aplicaciones todavía requieren programas rápidos y "ajustados" (requerimiento de poca memoria).

La *portabilidad del código fuente* es una característica de los lenguajes de programación y se puede interpretar de tres formas:

- El código fuente puede ser transportado de un procesador a otro y de un compilador a otro sin ninguna o muy pocas modificaciones.
- El código fuente permanece inalterado cuando cambia su entorno de funcionamiento por ejemplo cuando se instala una nueva versión de un sistema operativo.
- El código fuente puede ser integrado en diferentes paquetes de software sin que prácticamente se requieran modificaciones debidas a las características propias del lenguaje de programación.

Actualmente se habla ya de generaciones en los lenguajes de programación, al igual que en el hardware (ver figura 3.6). Sin embargo, no se comenzó a pensar en las diferentes generaciones de lenguajes de desarrollo de aplicaciones hasta que surgió el término *"lenguajes de cuarta generación"* (4GL), (de la misma forma que no se comenzó a pensar en la Primera Gran Guerra en Europa como la Primera Guerra Mundial, sino sólo hasta que apareció la segunda).

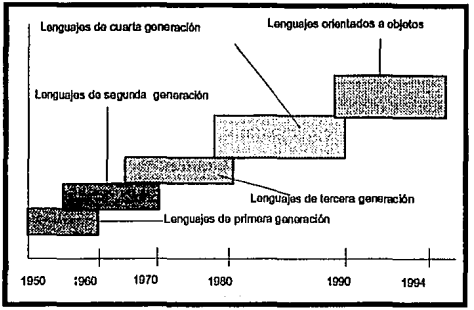


Figura 3.6 Lenguajes de programación

3.3.2 Primera generación de lenguajes

La primera generación de lenguajes de desarrollo de software se refiere al código máquina. Los ***lenguajes MAQUINA*** (ver figura 3.7) son aquellos que están escritos en lenguajes directamente entendibles por la máquina (computadora); en ellos las instrucciones son cadenas binarias (1's y 0's) que especifican una operación y las posiciones (direcciones) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina.

Las ***ventajas*** de programar en lenguaje MAQUINA son la posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior.

Los ***inconvenientes*** son:

- Dificultad y lentitud en la codificación.
- Poca fiabilidad.
- Los programas sólo son ejecutables en el mismo procesador (CPU).

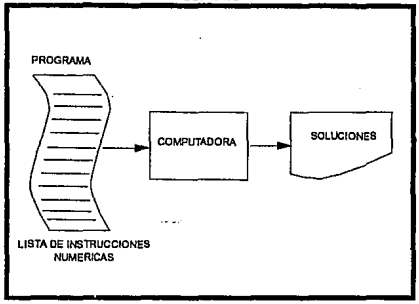


Figura 3.7 Lenguaje de MAQUINA

3.3.3 Segunda generación de lenguajes

La segunda generación de lenguajes fue desarrollada a finales de los 50's y principios de los 60's y ha servido como base de todos los lenguajes de programación modernos.

El lenguaje de segunda generación es el código *ensamblador* (ver figura 3.8); lenguajes procedurales de bajo nivel en los cuales cada instrucción del programa corresponde a una instrucción que el procesador puede realizar. Los lenguajes ENSAMBLADORES son dependientes del tipo de computadora asociado, ya que están diseñados para las características específicas y el conjunto de instrucciones de un determinado procesador. Su uso es básicamente para programar aplicaciones, tales como, sistemas operativos, pero no para el desarrollo de aplicaciones comerciales.

Un programa escrito en lenguaje ENSAMBLADOR no puede ser ejecutado directamente por la computadora en esto se diferencia esencialmente del lenguaje MAQUINA.

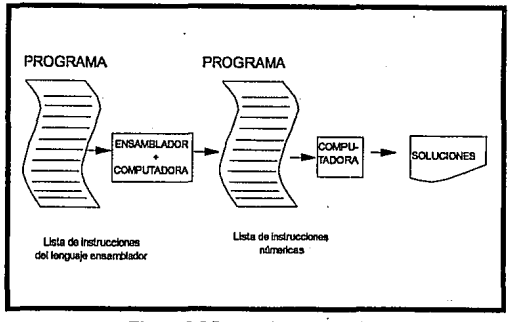


Figura 3.8 Lenguaje de ensamble

El lenguaje original escrito en lenguaje ENSAMBLADOR se denomina ***programa fuente*** y el programa traducido en lenguaje MAQUINA se conoce como ***programa objeto***, ya directamente entendible por la computadora.

El traductor de programas fuentes a objeto es un programa llamado ensamblador (assembler), existente en casi todas las computadoras.

La ***ventaja*** del lenguaje ENSAMBLADOR frente al lenguaje MAQUINA es que presenta mayor facilidad de codificación y en general, su velocidad de cálculo es similar.

Los ***inconvenientes*** son:

- La programación en código ensamblador requiere que el programador traduzca el diseño de su programa a secuencia de acciones de la máquina.
- Dependencia total de la máquina, lo que impide la transportabilidad de los programas (posibilidad de ejecutar un programa en diferentes máquinas).
- La programación en lenguaje ENSAMBLADOR tiende ha ser difícil y propensa a errores.

3.3.4 Tercera generación de lenguajes

Los lenguajes de alto nivel son los más utilizados por los programadores (ver figura 3.9). Están diseñados para que las personas escriban y entiendan los programas de un modo más fácil que los lenguajes MAQUINA y ENSAMBLADORES. Un programa escrito en un lenguaje de alto

nivel es independiente de la máquina, son ejecutados con poca o ninguna modificación en diferentes tipos de computadoras.

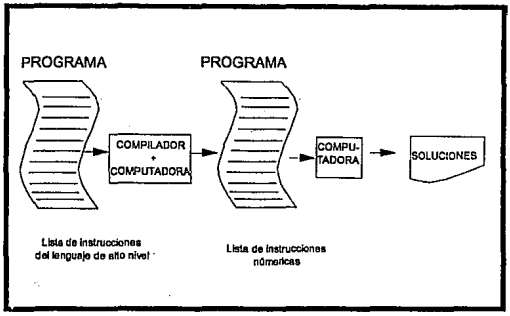


Figura 3.9 Lenguaje de alto nivel

Los lenguajes de alto nivel son llamados lenguajes de tercera generación y al igual que sucede con los lenguajes ENSAMBLADORES, los programas fuentes tienen que ser traducidos por los programas traductores, llamados en este caso compiladores e intérpretes, ejemplo de ellos: PASCAL, COBOL, BASIC y C. Estos lenguajes son procedurales, es decir que el programador tiene que especificar un procedimiento que el computador debe seguir para realizar una tarea determinada. Los traductores de lenguajes son programas que traducen a su vez los programas fuentes escritos en lenguajes de alto nivel a código máquina y se dividen en: **interpretes** y **compiladores** como a continuación se muestra en la figura 3.10.

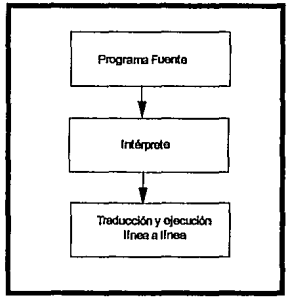


Figura 3.10 La interpretación de un programa

Un intérprete es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta. Un compilador es un programa que traduce los programas fuentes escritos en lenguajes de alto nivel a lenguaje MAQUINA.

Ventajas:

- El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos: como por ejemplo READ; WRITE; OPEN, etc.
- Reducción del costo de los programas.
- Transportabilidad.

3.3.5 Cuarta generación de lenguajes

A principio de los 80's surgió el término de cuarta generación para describir una nueva categoría de herramientas de programación interactiva usadas en el desarrollo de aplicaciones para Sistemas de Administración de Bases de Datos. Hoy en día estos lenguajes abarcan: lenguajes de consulta (Query) para usuario final, generadores de informes, de pantalla de entrada de datos y entornos completos de desarrollo, entre otros.

Los lenguajes de cuarta generación combinan características procedimentales y no procedimentales.

Desventajas:

- No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en los lenguajes MAQUINA y ENSAMBLADOR.
- Aumento de ocupación de la memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Correspondientes acciones (componente procedimental) mientras que pidiendo al mismo tiempo al usuario que indique el resultado deseado (componente no procedimental) para encontrar los detalles procedimentales aplicando su conocimiento del dominio específico.

3.3.6 Orientados a objetos

"Los lenguajes de programación orientados a objetos son lenguajes de programación no procedurales en los cuales los elementos del programador están considerados como objetos que pueden pasar mensajes a otros objetos. En un programa orientado a objetos, cada objeto tiene su propio código de programa, y es indispensable por sí mismo. El programa convierte al objeto en parte de un todo incorporado en una jerarquía de niveles".

Los lenguajes orientados a objetos son aquellos que permiten la estructuración de clases en jerarquías de herencia entre ellos se encuentran:

C++ (Lenguaje C con varias extensiones más "C con clases")

Es simplemente un lenguaje de programación orientado a objetos. Las características de orientación a objetos de C++ están interrelacionadas. En 1980, Bjarne Stroustrup cuando estaba trabajando en Bell Laboratories en Murray Hill, New Jersey, hizo varias extensiones al lenguaje C. Inicialmente se llamo "C con clases"; en 1983 el nombre se cambio a C++.

Las características de orientación a objetos, usando las palabras de Stroustrup permite programas estructurados con la claridad, extensibilidad y facilidad de mantenimiento sin perdida de eficiencia.

Aunque C++ se diseño inicialmente para ayudar en la gestión de procesos muy extensos, no está limitado a este uso. En realidad, los atributos de orientación a objetos de C++ pueden aplicarse con eficiencia virtualmente a cualquier tarea de programación. No es raro ver usar C++ para proyectos tales como un editor, bases de datos, sistemas de archivo personales y programas de comunicaciones. También pueden construirse sistemas de software de alto rendimiento, ya que C++ comparte la efectividad de C.

ADA (Lady "Ada" Lovelace-Ausgusta Ada Byron)

El lenguaje de programación ADA es un descendiente reciente de PASCAL. ADA incorpora muchos conceptos de PASCAL y de otros lenguajes modernos. Siguiendo una intensiva competencia de diseño patrocinada por el Departamento de Defensa de los Estados Unidos, el lenguaje GREEN, desarrollado por J. Ichbiah y colegas del grupo CII-Honeywell-Bull en Francia, se seleccionó para ser el lenguaje de programación ADA (ADA 83). ADA recibe el nombre por Lady Ada Lovelace quién fue asociada de Charles Babbage durante el siglo pasado. Ella es ampliamente reconocida como la primera programadora de computadora.

El lenguaje ADA se desarrolló bajo patrocinio del Departamento de Defensa de los Estados Unidos para apoyar el desarrollo de productos de la programación para sistemas de computación "empotrados"[8].

ADA no sólo incorpora características especiales para sistemas empotrados (manejo de interrupciones externas, acceso a detalles de representación de datos, bajo nivel de entrada-salida), sino que también proporciona muchas características avanzadas para apoyar la programación en una amplia variedad de aplicaciones. Características tales como el encapsulado de datos y un mecanismo de concurrencia hacen de ADA un lenguaje de interés para una comunidad más amplia que el área de sistemas empotrados para la cual fue desarrollada.

LISP (Lenguaje de Proceso de Listas)

Es un lenguaje especialmente adecuado para la manipulación de símbolos y el procesamiento de listas en problemas combinatorios. Usado casi exclusivamente por la comunidad de inteligencia artificial, el lenguaje es especialmente adecuado para la prueba de teoremas, para búsquedas en árboles y otras actividades de resolución de problemas. Los subprogramas están implementados como funciones que hacen un gran uso de la recursividad. Debido a que cada función de LISP es una unidad independiente, se puede conseguir una gran reusabilidad mediante la creación de bibliotecas de funciones primitivas.

---

[8] Un sistema de computación empotrado es un componente de un sistema mayor y provee funciones de computación, comunicación y control para ese sistema. Los sistemas empotrados a menudo son instrumentados utilizando microprocesadores y con frecuencia incorporan concurrencia y procesamiento en tiempo real guiado por interrupciones.

PROLOG

Es un lenguaje de programación que se ha usado mucho en la construcción de sistemas expertos. Al igual que LISP, PROLOG proporciona medios de soporte para la representación del conocimiento. Dentro del lenguaje se usa una estructura de datos uniforme, denominada término, para construir todos los datos y todos los programas. Cada programa consiste en un conjunto de cláusulas que representa hechos, reglas e inferencias. Tanto en LISP como PROLOG son especialmente adecuados para los problemas que tratan objetos y sus relaciones. Por esta razón, se dice que LISP y PROLOG son lenguajes orientados a objetos.

SMALLTALK (Lenguaje de pocas palabras)

Uno de los primeros lenguajes realmente orientados a objetos que introduce una estructura de datos y control radicalmente diferente de los lenguajes de programación convencionales. SMALLTALK permite al programador definir ***objetos*** que combinan una estructura de datos y conjunto de ***métodos*** (subprogramas). En la actualidad dentro del mercado, se considera que la aproximación orientada al objeto liderada por SMALLTALK puede llevar a la creación de componentes de programas reutilizables que reducirían enormemente el tiempo de desarrollo (y el tamaño, medido en líneas de código) para los grandes sistemas de software. Como ADA, SMALLTALK es más que un lenguaje de programación. Se ha creado un entorno para él que ayuda en el desarrollo de programas.

Los elementos básicos de los lenguajes orientados a objetos se presentan a continuación (ver figura 3.11).

ELEMENTOS DE LOS LENGUAJES ORIENTADOS A OBJETOS	ABSTRACCION
	HERENCIA
	POLIMORFISMO
	ENCAPSULACION
	TIPIFICACION
	SOBRECARGA
	LIGA DINAMICA
	FUNCIONES DIFERIDAS
	GENERALIZACION
	ASIGNACION DINAMICA
	ESTANDARES

Figura 3.11 Elementos de los lenguajes orientados a objetos

- **Abstracción de datos:** la base del diseño orientado a objetos es la construcción de sistemas como colecciones estructuradas de tipos de datos abstractos. Un tipo de dato abstracto define datos conjuntamente con las operaciones que los manipulan, como ejemplo de este tipo de datos se encuentran los números enteros.
  La abstracción consiste en ocultar los detalles irrelevantes para la comprensión manipulación del tipo.
  Los lenguajes de programación que apoyan la abstracción de datos permiten agrupar en la misma declaración los datos y las operaciones que los manipulan, ocultando los detalles de implementación. Tradicionalmente, en los lenguajes de programación como PASCAL, por ejemplo, se declaran por separado los datos y las operaciones, y no existe la posibilidad de ocultar la representación concreta elegida para los valores.

- **Herencia:** es una relación entre clases que permite declarar una clase subclase. Los lenguajes orientados a objetos presentan dos variantes básicas: herencia sencilla y herencia múltiple, en la segunda, una clase puede heredar más de una clase.

- **Tipificación:** la clasificación de objetos en tipos se genera naturalmente en cualquier dominio, para categorizar objetos de acuerdo a su uso y comportamiento. Un tipo determina las características estructurales y de comportamiento de los objetos que le pertenecen. El objetivo de un sistema de tipos es estructurar el universo de objetos de tal forma que se regule su uso en diversos contextos.

- **Polimorfismo:** en los lenguajes de programación el polimorfismo se genera al admitir que una expresión o valor toma más de un tipo. Esto permite definir funciones que son aplicables a objetos de diferentes tipos.

- **Sobrecarga:** se refiere a la posibilidad de definir varias funciones con el mismo nombre. En particular permite utilizar la misma sintaxis para los operadores aritméticos, como el símbolo + que está definido para más de un tipo de datos.

- **Liga Dinámica:** permite asociar el significado de una función al momento de su activación, en contraposición de la liga estática que se genera al momento de la compilación.

- **Funciones Diferidas:** se especifican en una clase y su implantación se pospone para ser provista por los descendientes de la clase. Este tipo de funciones sólo tiene sentido si se trabaja en conjunto con la liga dinámica. Cuando una clase tiene una o más funciones diferidas se le denomina abstracta. Una clase abstracta no puede ser instanciada; es decir; no se pueden generar objetos de esa clase.
  Las funciones diferidas sirven para especificar servicios que deben proporcionar todos los descendientes de una clase. Son una herramienta valiosa para modelar conceptos abstractos de diversas maneras.

- **Generecidad:** implementar clases parametrizadas. Por ejemplo si se tiene implantado un módulo para manipular listas de enteros y se desea aplicar a listas de caracteres, es necesario crear una nueva versión de la clase. Una solución a este problema sería copiar la clase que almacena enteros y reemplazar con un editor de textos todas las ocurrencias de la cadena entero por la cadena carácter. Esta solución puede conducir a errores fácilmente. Una alternativa sería crear una clase que pueda recibir como parámetro el tipo de los datos de almacenar en listas y especializarla de acuerdo a los requerimientos.

- **Alojamiento dinámico de memoria:** debido a la forma dinámica con que se crean y eliminan objetos durante la ejecución de un sistema orientado a objetos, es necesario contar con mecanismos que optimicen el uso de los recursos de memoria disponibles. Los mecanismos de recolección de basura liberan automáticamente los segmentos de memoria utilizados por objetos que ya no son referenciados.

- **Esforzamiento de principios:** dependiendo de las características particulares de su diseño, cada lenguaje esforza en mayor o menor medida el uso de los principios de la tecnología orientada a objetos cuando se utiliza para el desarrollo de sistemas. Entre las características particulares de un lenguaje que influye en este sentido, se encuentran aspectos como la medida en que se apoya el uso de abstracciones y la seguridad de las mismas, el poder de sus mecanismos de herencia, el uso de liga dinámica, etc.

- **Estándares:** se utiliza para regular las características de un lenguaje en particular y garantizar la compatibilidad de distintos compiladores del mismo, habilitando la portabilidad entre distintas plataformas.

3.4 BASES DE DATOS ORIENTADAS A OBJETOS

Los sistemas administradores de bases de datos orientadas a objetos nacieron en el inicio de los años 80's. El primer sistema administrador de base de datos orientada a objetos estuvo disponible en 1987 con la introducción de GEMSTONE, de Servio Corporation (entonces Servio Logic), y Vbase, predecedor de Ontos, de Ontos Inc. (antes Ontologic). Hoy en día existen al menos 25 productos de sistemas administradores de bases de datos orientados a objetos en el mercado o en curso de desarrollo y de próxima disponibilidad.

3.4.1 Definición de un sistema administrador de base de datos orientada a objetos

Hoy en día no existe una definición estándar de un sistema administrador de base de datos orientada a objetos, sin embargo se ha establecido un primer consenso para su definición, descrito a través de las "doce reglas de oro".

Los sistemas administradores de bases datos orientadas a objetos almacenan **objetos** no sólo datos. Objetos que representan, además, entidades u objetos del mundo real que están siendo modelados en la aplicación. Objetos en el sentido de combinaciones encapsuladas de estructuras de datos (atributos, propiedades) y procedimientos asociados (métodos) que descubren su comportamiento.

3.4.2 Características de un sistema administrador de base de datos orientada a objetos

Las reglas de oro[9], permiten caracterizar un sistema administrador de base de datos orientada a objetos, las primeras cinco, por sus funcionalidades en tanto a un sistema administrador de base de datos y las siete restantes por su aplicación del paradigma como se observa en la figura 3.12.

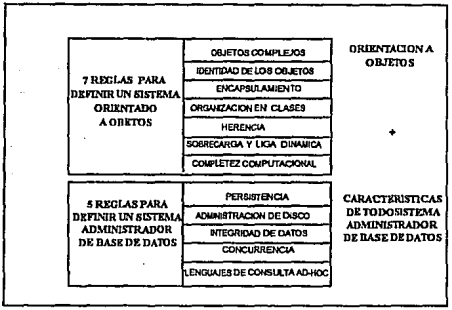


Figura 3.12 Reglas de oro de un sistema administrador de base de datos orientada a objetos=orientación a objetos+sistema administrador de base de datos

Las características de un sistema administrador de una base de datos orientada a objetos provenientes del modelo de datos orientados a objetos se dan a continuación considerando su adaptación a las necesidades planteadas por el enfoque de los sistemas administradores de base de datos.

- **Persistencia:** en un lenguaje de programación, todos los objetos manipulados desaparecen al término de la ejecución del programa que los creó y/o manipuló. Por el contrario en un sistema administrador de base de datos todos los objetos son creados para persistir y poder ser consultados y/o manipulados posteriormente por la aplicación.

---

[9] M. Atkinson et al. "The Object-Oriented Database System Manifesto", Proceedings of the 1st Intl. Conf. on Deductive and Object-Oriented DataBases, Kyoto, Japan, december, 1989.

- **Administración del disco:** los avances tecnológicos alcanzados en los dispositivos de almacenamiento permiten almacenar cada vez mayores volúmenes de datos a menor costo. Los sistemas administradores de bases de datos son capaces de asegurar el acceso eficiente a los datos almacenados.

- **Integridad de los datos:** la integridad se define como la propiedad que asegura la calidad de los datos, respetando las propiedades o reglas especificadas en su definición. La calidad global de las bases de datos incluye aspectos como: integridad semántica (fidelidad en la descripción de los objetos del mundo real), seguridad en el acceso y seguridad en el funcionamiento. Todos estos aspectos que constituyen la integridad de la base de datos, requieren de adaptación al modelo orientado a objetos, planteando problemas tales como: especificación de reglas sobre los objetos, definición de protecciones y autorizaciones de acceso sobre los objetos.

- **Concurrencia en el acceso:** una de las principales ventajas que ofrece la administración de los datos por un sistema administrador de base de datos es la posibilidad de poner a disposición de varios usuarios, simultáneamente la misma fuente de información. La noción de transacción que asegura la ejecución de un conjunto de operaciones de manera atómica debe ser adaptada para aplicarse sobre los objetos administradores por el sistema administrador de base de datos orientado a objetos, permitiendo la ejecución y control de transacciones "largas".

- **Lenguaje de consulta:** la principal operación solicitada por los usuarios y sus aplicaciones es la consulta de la información previamente almacenada. Tomando en consideración lenguajes de consulta como **SQL**, es necesario seguir ofreciendo lenguajes de este tipo, que tomando en consideración el modelo orientado a objetos, permita extraer objetos o parte de ellos a través de un lenguaje sencillo.

3.4.3 Ventajas de los sistemas administradores de bases de datos orientadas a objetos

Una de las principales cualidades del paradigma orientación a objetos aplicado a las bases de datos es su enfoque para el modelado del esquema conceptual, reduciendo la distancia semántica entre los objetos reales, los objetos representados y los objetos almacenados.

Es de gran utilidad para la persona que al programar se cuente con un ambiente de desarrollo que permita la representación de entidades del mundo real, como objetos de la base de datos y facilite su manipulación sin necesidad de separarlos para ser almacenados y reconstruirlos en el momento de ser consultados (ver figura 3.13).

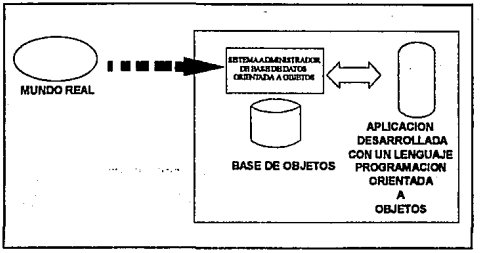


Figura 3.13 Ambiente integrado del desarrollo

Los sistemas administradores de bases de datos orientadas a objetos pueden reducir gradualmente la cantidad de código de las aplicaciones esto es, suprimiendo las funcionalidades (operaciones) de los programas de aplicación e incorporándolas en torno de los objetos (a través de los métodos).

La posibilidad de reusabilidad del código, así como la presencia de buenas librerías de clases, permite a los desarrolladores de aplicaciones ser más productivos.

3.4.4 Costos actuales de los lenguajes y bases de datos orientados a objetos

A) VISUALWORKS (Release Versión 1.0 para Windows incluye SMALLTALK)

Características

Es un ambiente de desarrollo de aplicaciones que incluye el poder del lenguaje SMALLTALK, junto con constructores de interfaces gráficas orientadas a objetos hacia base de datos externas. El lenguaje SMALLTALK es el fundamento de VISUALWORKS. La biblioteca de clases de SMALLTALK proporciona los bloques constructores esenciales para el desarrollo de aplicaciones, y contiene más de 650 clases y más de 14,000 métodos todos ellos para utilizarse. VISUALWORKS es un conjunto de herramientas de desarrollo integradas que auxilian a los desarrolladores a encontrar, entender y reutilizar código, con el objeto de maximizar la productividad de los programadores proporcionando, además, los beneficios de la tecnología orientada a objetos

Su interfaz a bases de datos externas permite combinar el poder de las bases de datos jerárquicas y relacionales con tecnología de programación orientada a objetos, con el objeto de desarrollar aplicaciones cliente/servidor. VISUALWORKS proporciona acceso directo a ORACLE y SYBASE, VISUALWORKS cuenta con EXDI (External Database Interfaz).

Plataforma

VISUALWORKS con su característica de recopilación automática, ofrece portabilidad instantánea a través de PC's (con MS-Windows 3.1 y OS/2 2.0), Macintosh y plataformas UNIX.

Precio en DLS **$5,829**

B) ISE EIFFEL 3

Características

Es un ambiente de programación que integra componentes que en conjunto proporcionan una buena alternativa para desarrollar sistemas para plataforma UNIX en el sector productivo. Sus componentes son EIFFEL Bench (incorpora al compilador del lenguaje una tecnología que se combina compilación e interpretación), EIFFEL Base (es la biblioteca de clases básicas de EIFFEL), EIFFEL Vision (proporciona funcionalidades avanzadas de graficación) EIFFEL Build y EIFFEL Store (soporta una amplia gama de Sistemas Administradores de Bases de Datos Relacionales y Orientados a Objetos).

Plataforma

UNIX

Precio en DLS **$2,915**

C) GEMSTONE

Características

Es un sistema administrador de base de datos de objetos que proporciona un lenguaje de definición de datos (DDL) y de manipulación de datos (DML). En GEMSTONE los datos pueden

acceder simultáneamente desde SMALLTALK, C++ y C; así como por lenguajes que soportan llamadas a funciones en C tales como ADA, COBOL, FORTRAN, LISP y OBJETIVE C. GEMSTONE consiste de dos componentes de software: GEM, que accede y ejecuta objetos, y STONE, que administra la base de datos.

Plataforma

Para GEMSTONE se puede utilizar una IBM RS/6000, DEC VAX, DECstation, HP 9000 serie 700, o SUN SPARCstation, entre muchas otras.

Precio en DLS **$1,155**

D) VISUAL C++ (Versión 1.0)

Características

Provee facilidades al programador para realizar abstracción de datos, proyectando mecanismos que permiten hacer programación orientada a objetos. Como se basa en un lenguaje tradicional, en C++ no es necesario el uso de objetos.

Precio en DLS **$499**

E) VISUAL BASIC PROFESIONAL (Versión 3.0)

Características

Las posibilidades que nos presenta el ambiente de desarrollo de VISUAL BASIC 3.0 de Microsoft son muy variadas. Sin embargo, tal como sucede al iniciar el uso de una nueva herramienta, el programador suele posponer de manera constante la inclusión de técnicas o características de complejidad media por temor a meterse en terrenos resbalosos.

Trabaja en ambiente Windows, lo que hace que sus aplicaciones sean amigables para el usuario. El VISUAL BASIC no es un manejador de base de datos sino una herramienta de desarrollo. VISUAL BASIC requiere de un manejador de base de datos (por ejemplo: ACCESS).

Precio en DLS **$495**

F) PARADOX FOR WINDOWS (Versión 1.0)

Características

Debido al ambiente de Windows con que trabaja sus aplicaciones se hacen amigables para los usuarios, y para los programadores es un conjunto de herramientas de desarrollo integradas que auxilian a los desarrolladores a encontrar, entender y reutilizar el código (ver Apéndice A).

Plataforma

PARADOX corre bajo Windows en una computadora personal compatible con IBM. El equipo mínimo es:

- Microprocesador 80286
- RAM 4MB
- Disco duro 20 MB
- Sistema de Video EGA (CGA no soportado)
- Mouse compatible con Windows
- Impresora

Precio en DLS **$150**

COTIZACIONES DE PAQUETES ORIENTADOS A OBJETOS

COTIZACION: VISUALWORKS PARCRANGER

Cant.	Producto	Descripción	Precio
1	VisualWorks ParcRanger Unix	VisualWorks R1.0 el cual incluye: ObjectWorks/Smalltalk R4.1 es un Ambiente de Desarrollo de Aplicaciones en Smalltalk para una de las siguientes plataformas Unix: DEC MIPS, HP 9000/7XX, IBM RS/6000, SEQUENT, Sun SPARCstation.	$2,015.00 DLS
		Extención del ParcRanger para que cubra otra marca dentro de la misma plataforma Unix.	$1,155.00 DLS.
1	VisualWorks ParcRanger PC	VisualWorks R1.0.el cual incluye: ObjectWorks/Smalltalk R4.1 es un Ambiente de Desarrollo de Aplicaciones para computadora personal con uno de los siguientes Sistemas Operativos: MAC OS v7, OS/2 v2, Windows 3.1.	$2,915.00 DLS
		Extensión del ParcRanger para que cubra otro Sistema Operativo dentro del rango de Computadoras Personales.	$1,155.00 DLS.

Características de ParcRanger

- Incluye un Advanced Programming ObjectKit y un C Programming ObjectKit.
- Licencia para reproducir ilimitadamente software y manuales para uso académico de: profesores, investigadores y alumnos durante la extensión de la licencia, por marca de plataforma elegida.
- Actualizaciones automáticas de software durante la extensión de la licencia.
- Licencia Anual.

micro alven, s.a. de c.v.
COMPUTADORAS ACCESORIOS [illegible]

México, D.F., a 1 de Marzo de 1993.

PEMEX EXPLORACION Y PRODUCCION
AT'N. ING. GERARDO GUZMAN DOMINGUEZ

En respuesta a su atenta solicitud hacemos la presentacion de la siguiente cotizacion:

TURBO C++ FOR WINDOWS V 3.1 VISUAL EDITION	200 US
VISUAL BASIC 2.0 FOR WINDOWS	177 US
PARADOX FOR WINDOWS	150 US

NOTAS:
Los precios anteriores se encuentran cotizados en dolares y se convertiran al tipo de cambio del dia que se efectue la operacion. No incluyen el 10% de I.V.A. y se encuentran sujetos a cambio sin previo aviso.

Sin mas por el momento y en espera de vernos favorecidos con su preferencia, quedamos de Uds.

A T T E.

ING. ENRIQUE COSTA [illegible]

Teix, Sistemas de Información Estratégica, S.A. de C.V.
Antonio Anza 43 Col. Roma México, D.F. 06700
Tels: 564-7146 264-0375 Fax. 264-6539

COTIZACIÓN

PRODUCTOS

Cantidad	Producto	Descripción	Precio
1	Visual Works Release	Versión 1.0 para Windows, incluye SmallTalk	$ 5,829 DLS
1	Visual Basic Profesional	Versión 3.0	$ 495 DLS
1	Visual C++	Versión 1.0	$ 499 DLS
1	Paradox	Versión 3.5	$ 875 DLS

CONDICIONES COMERCIALES

Precios

Se han reducido deacuerdo a la fracción arancelaria para software indicada en el Tratado de Libre Comercio para NorteAmérica. No incluyen el impuesto al valor agregado y para su conversión se tomará el tipo de cambio vigente a la fecha en que se efectué el pago. Pueden variar sin previo aviso.

Forma de Pago

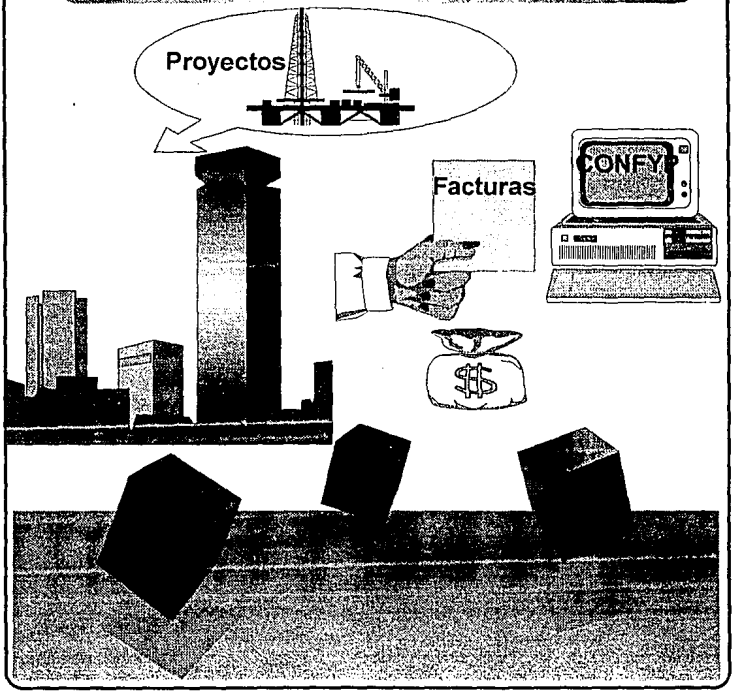
Se debe cubrir 50% al poner la orden de compra y 50% al aviso de envio de productos.

Gastos

Los gastos son estimados y se ajustarán al momento de la importación.

CAPITULO IV

Software de Aplicación



SOFTWARE DE APLICACION

Debido a que en muchos procesos administrativos se requieren realizar constantemente y en forma repetitiva una serie de pasos, para alcanzar el objetivo que se persigue, y dado que estos pasos son mecánicos y susceptibles a la automatización, es recomendable el uso de sistemas de cómputo para obtener una mayor rapidez, eficiencia y veracidad en la realización de dichos procesos.

4.1 INTRODUCCION

Todo sistema de software basa su existencia en el cumplimiento de un mecanismo de desarrollo, este mecanismo puede ser susceptible a cambios; es decir, existen en la actualidad varios mecanismos para un desarrollo de software, la elección de uno u otro depende realmente del programador o en su defecto del analista, los cuales pueden seleccionar el más eficiente.
Tomando en cuenta estos aspectos y considerando que el tema a tratar es la metodología de análisis y diseño orientada a objetos se considera el mecanismo orientado a objetos para realizar el sistema de Control de Facturas y Proyectos con el fin de que de alguna forma conceptual y operativa se especifiquen las ventajas y desventajas de este mecanismo.

4.2 ANALISIS

A partir de este momento y tomando en cuenta los conceptos y fases de la metodología de análisis y diseño con orientación a objetos se inicia el análisis para el sistema de Control de Facturas y Proyectos, (Ver capítulo II).

4.2.1 Definición del problema

La Gerencia de Desarrollo Tecnológico (Pemex) se encarga de controlar todos los proyectos que realizan para Pemex, diferentes Empresas y que tienen a su cargo las Subgerencias correspondientes. El control interno de estos proyectos implica el conocimiento de ciertos procesos que inician desde que se propone un proyecto, se acepta, se desarrolla, se modifica según sean las necesidades y se da por terminado. Para llevar acabo este control es necesario tener disponible determinada información para la elección de ciertos criterios ejecutivos, además de generarse un proceso de control de facturación atribuible a estos proyectos en forma actualizada.

En la actualidad en esta Gerencia, existe la necesidad de automatizar los procesos de Control de Proyectos y Facturas. Con el objeto de apoyar estos procesos, la Dependencia mencionada encargó al Instituto Mexicano del Petróleo la creación del **SISTEMA DE CONTROL DE FACTURAS Y PROYECTOS** para realizar esta tarea con eficiencia y rapidez.

4.2.2 Objetivo

El sistema de Control de Facturas y Proyectos cumplirá con los requerimientos del control de proyectos y facturas de la Gerencia de Desarrollo Tecnológico (Pemex) pero de forma automatizada, tomando en cuenta los criterios antes mencionados. Además el sistema proporcionará al usuario informes ejecutivos que permitan establecer criterios presupuestales en un futuro, considerando también la consulta rápida de los proyectos y facturas. El flujo de información del sistema correrá desde las Empresas que tienen a su cargo los proyectos y facturen, hasta llegar a la Gerencia de Desarrollo Tecnológico como se representa a continuación en la figura 4.1.

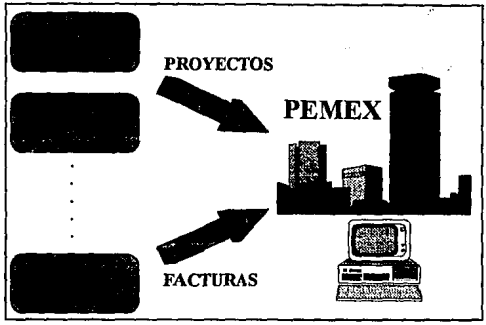


Figura 4.1 Diagrama de flujo de información

4.2.3 Procesos de control

El proceso de control de proyectos y facturas contiene una secuencia de pasos a seguir como son:

1.- **Autorización del proyecto, momento en el cual se describen todas las especificaciones:**

Clave de proyecto	Clave que se asigna a un proyecto en particular.
Nombre de proyecto	Descripción del proyecto mismo.
Tipo de proyecto	Pemex clasifica a sus proyectos de acuerdo a los objetivos de los mismos. El tipo de proyecto esclarece esta clasificación.
Estado de proyecto	Los proyectos pueden estar en diferentes estados (Autorizados, Terminados, Cancelados, etc.).
Renglón del gasto	En Pemex clasifican a los proyectos de acuerdo a si son de inversión o de operación. [208 o 308]
Gerencia	Gerencia la que es responsable del proyecto.
Subgerencia	Subgerencia la que es responsable del proyecto.
Empresa	Empresa responsable del proyecto.
Fecha de inicio	Fecha programada en que inicia el proyecto.
Fecha de término	Fecha programada en que termina el proyecto.
Costo total	Costo del proyecto.

2.- Asignación presupuestal anual para el desarrollo del proyecto.

Presupuesto	Cantidad anual asignada a un proyecto.

3.- Asignación de un responsable del Pemex del proyecto.

Responsable	Persona responsable del cumplimiento del proyecto.

4.- Facturación atribuible a un determinado proyecto.

Clave de la factura	Clave que se le asigna a una factura en particular.
Clave del proyecto	Clave que se le asigna a un proyecto en particular.
Fecha de elaboración de la factura	Es la fecha de elaboración de la factura.
Periodo	Es el periodo que ampara a la factura.
Estado de la factura	Estado en el cual se encuentra la factura (Autorizada, Devuelta, Recibida y Pendiente).
Total	Cantidad erogada por la factura.
Concepto	Justificación de la factura.

a) Proceso de estado de la factura desde su llegada a la Gerencia de Desarrollo Tecnológico hasta su autorización

Ubicación de la factura	Lugar en donde la factura se encuentra en su proceso de autorización.

Una vez identificado el problema y sabiendo los procesos de control de proyectos y facturas los cuales están ampliamente ligados, se procede a describir los requerimientos de información para el sistema:

PROYECTO
Clave del proyecto
Tipo de proyecto
Nombre del proyecto
Estado del proyecto
Renglón del gasto
Gerencia
Subgerencia
Empresa
Fecha de inicio
Fecha de término
Costo
Presupuesto
Responsable de proyecto

Cada proyecto tiene asociadas un cierto número de facturas, para contabilizar el monto de las mismas y relacionarlo con el presupuesto asignado. Las facturas contienen la siguiente información:

FACTURA
Clave de factura
Clave de proyecto
Fecha de elaboración
Periodo
Estado de la factura
Total
Concepto
Fecha de entrada a una Dependencia
Fecha de salida de una Dependencia
Dependencia

4.2.4 Funcionalidades

El sistema a desarrollar debe proporcionar al usuario las siguientes *funcionalidades:*

FUNCIONALIDADES
Dar de alta, modificar y dar de baja un proyecto
Dar de alta, modificar y dar de baja un estado de proyecto
Dar de alta, modificar y dar de baja un tipo de proyecto
Dar de alta, modificar y dar de baja una Empresa
Dar de alta, modificar y dar de baja una Gerencia de Pemex
Dar de alta, modificar y dar de baja una Subgerencia de Pemex
Dar de alta, modificar y dar de baja un responsable del proyecto
Dar de alta, modificar y dar de baja un presupuesto anual
Dar de alta, modificar y dar de baja la ubicación de la factura
Dar de alta, modificar y dar de baja una factura
Dar de alta, modificar y dar de baja una Dependencia
Generar informes ejecutivos donen se proporcione información presupuestal y información relacionada con los proyectos y facturas: a)Datos concernientes a un proyecto determinado con sus respectivas facturas. (Control financiero de proyectos) b)Datos concernientes una factura determinada. (Control financiero de facturas) c)Las facturas recibidas en la Gerencia de Desarrollo Tecnológico por un periodo determinado. (Control de facturas) d)Genera una consulta de la ubicación de todas las facturas en un periodo de tiempo determinado.(Estado actual de facturas) Por Dependencia y Total. e)Proporciona la consulta de todos los proyectos desarrollados en la Gerencia de Desarrollo Tecnológico por un periodo determinado y (Estado financiero de proyectos): Por Empresa, Gerencia,Subgerencia , Empresa y Gerencia, Empresa y Subgerencia, Gerencia y Subgerencia, Empresa Gerencia y Subgerencia, y Total.

4.2.5 Candidatos para clases y métodos

Analizando los requerimientos del sistema y utilizando el método de análisis orientado a objetos de Booch se obtuvieron los siguientes candidatos:

CANDIDATOS PARA CLASES	CANDIDATOS PARA METODOS
Proyectos	Crear,cancelar,modificar
Tipos de proyecto	Crear,cancelar,modificar
Estados de proyecto	Crear,cancelar,modificar
Empresas	Crear,cancelar,modificar
Gerencias	Crear,cancelar,modificar
Subgerencias	Crear,cancelar,modificar
Responsables	Crear,cancelar,modificar
Presupuestos	Crear,cancelar,modificar
Facturas	Crear,cancelar,modificar
Estados de factura	Crear,cancelar,modificar
Ubicaciones de la factura	Crear,cancelar,modificar
Dependencias	Crear,cancelar,modificar

4.3 DISEÑO

Una vez que se define el problema y de alguna manera se especifican los requerimientos funcionales del sistema y considerando la información que deberá contener el sistema, se pasa a un punto que es fundamental dentro del marco de creación de software, como es el diseño. Donde se especifican en forma esquemática las clases y métodos utilizados y contenidos dentro del sistema. Se puede considerar en este sistema que los objetos elegidos considerando la orientación a objetos son el proyecto y la factura, basándose en este hecho se consideran las siguientes clases:

En el contexto general del sistema se considera un diagrama de clases general del cual dependen otras clases. Para el desarrollo del sistema control de facturas y proyectos el diagrama de clases general propuesto se muestra a continuación en la figura 4.2.

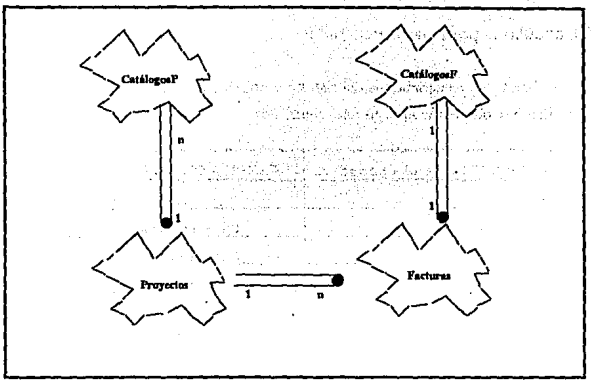


Figura 4.2 Diagrama general de clases del sistema CONFYP

Como se observa en la figura 4.2 se consideran dos clases de catálogos llamados uno CatálogosP y otro CatálogosF para marcar la diferencia entre los catálogos requeridos por la clase proyectos y los catálogos requeridos por la clase facturas; aunque en realidad se podrían considerar en una sola clase llamada Catálogos.

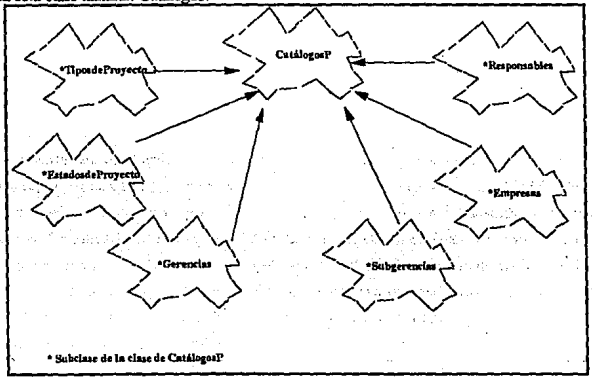


Figura 4.3 Diagrama de clase CatálogosP y subclases

La clase CatálogosP se muestra en la figura 4.3 y como se observa contiene todos los catálogos relacionados con la clase proyectos y que en un momento dado apoyan a esta clase.

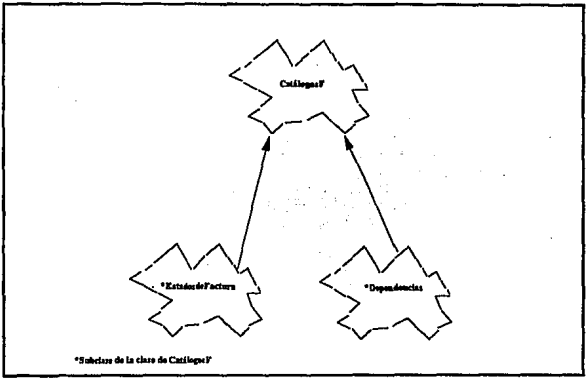


Figura 4.4 Diagrama de clase CatálogosF y subclases

La clase CatálogosF mostrada en la figura 4.4 contiene todos los catálogos relacionados con la clase Facturas que en un momento dado apoyan a esta clase. A continuación se presenta la figura 4.5 con la clase Proyectos.

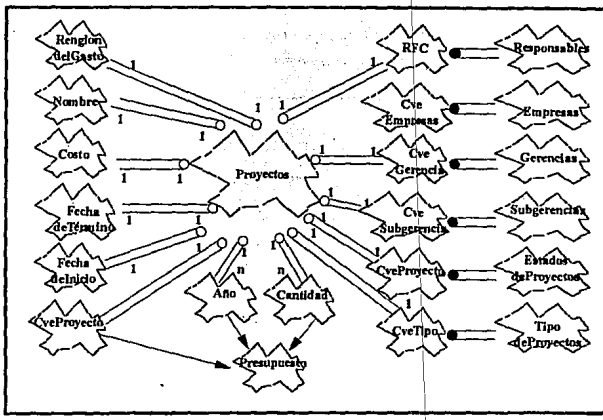


Figura 4.5 Diagrama de clase proyectos

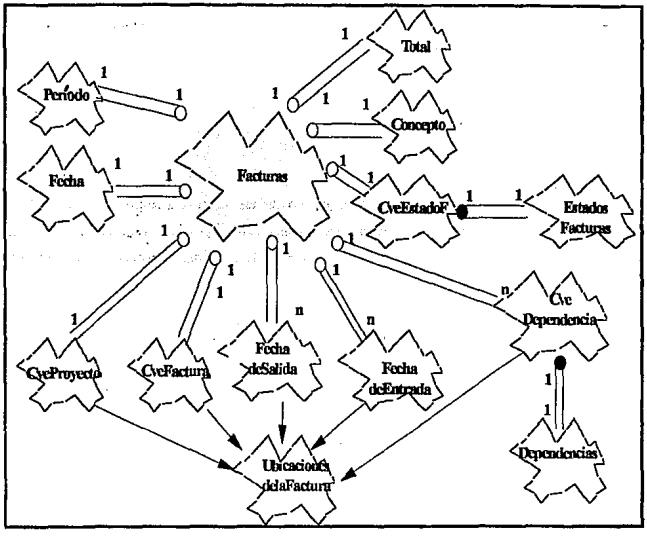


Figura 4.6 Diagrama de clase facturas

La clase facturas se representa en la figura 4.6.

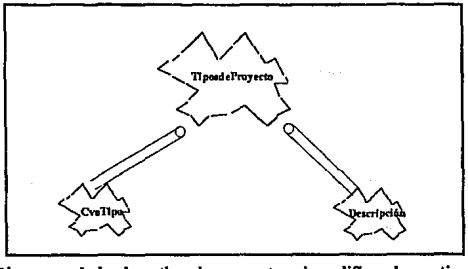


Figura 4.7 Diagrama de la clase tipo de proyectos ejemplificando un tipo de catálogo

Una vez definido el análisis y diseño del sistema se entra a la fase denominada evolución, la cual consiste en el desarrollo del software.

4.4 EVOLUCION

Generalmente al realizar el análisis y diseño de un sistema siempre se contempla el software que se emplea en el desarrollo del mismo, ya que de ello depende en gran parte el éxito del sistema. Lo anterior denota que el análisis y diseño orientado a objetos descrito anteriormente se elabora en base al software que se emplea.

4.4.1 Justificación del software empleado

Considerando que el método de análisis y diseño que se emplea, para la realización de este sistema como un método orientado a objetos, se vio la necesidad de emplear para el desarrollo de software, un manejador de base de datos orientado a objetos con la finalidad de esclarecer y verificar hasta cierto punto las características de este método. Cabe mencionar que es válido realizar un análisis y diseño orientado a objetos y no emplear ya sea un manejador de base de datos o un lenguaje de programación orientado a objetos, sin embargo se recomienda utilizarlos, para el mejoramiento y eficiencia del software. El manejador de base de datos empleado es **"PARADOX FOR WINDOWS"** debido a las características mostradas en el capítulo III punto 3.4.4 y a las características externas mostradas a continuación.

4.4.1.1 Características externas

1.- **Desarrollo de software relativamente más rápido, que con un lenguaje de programación orientado a objetos.**
2.- **Un ambiente de trabajo fácil y amigable para el usuario.**
3.- **Presentación aceptable.**
4.- **Manejador de base de datos.**
5.- **Costo**
6.- **Facilidad de acceso**

4.4.2 Estructuras y nombres de tablas en PARADOX del sistema

En la figura 4.8 se muestran los nombres de las tablas en PARADOX empleadas y los nombres comunes de las tablas para un mejor manejo.

NOMBRE COMUN	NOMBRE DE LA TABLA
DEPENDENCIAS	DEPENDEN.DB
ESTADOS DE LA FACTURA	EDOFACT.DB
ESTADOS DEL PROYECTO	EDOPROY.DB
EMPRESAS	EMPRESAS.DB
GERENCIAS	GERENCIA.DB
RESPONSABLES	RESPONSA.DB
SUBGERENCIAS	SUBGEREN.DB
TIPOS DE PROYECTO	TIPOS.DB
PROYECTOS	PROYECTO.DB
PRESUPUESTOS	PRESUPUE.DB
FACTURAS	FACTURAS.DB
UBICACIONES DE LA FACTURA	FECHFACT.DB

Figura 4.8 Nombre de tablas y nombre común

A continuación se muestran las estructuras de todas las tablas empleadas en el sistema incluyendo tanto las tablas de tipo catálogos como las de captura, se utiliza el nombre común de la tabla para identificarla.

TABLAS CATALOGOS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_DEPENDENCIA	A	6	*	Clave que identifica una dependencia
DEPENDENCIA	A	45		Descripción de la dependencia

DEPENDENCIAS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_ESTADOF	A	1	*	Clave única del estado de la factura
EDO_FACTURA	A	10		Descripción del estado de la factura

ESTADOS DE FACTURA

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_ESTADOP	A	1	*	Clave única del estado del proyecto
EDO_PROYECTO	A	10		Descripción del estado del proyecto

ESTADOS DE PROYECTO

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_EMPRESA	A	6	*	Clave única de la Empresa
EMPRESA	A	45		Descripción de la Empresa

EMPRESAS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_GERENCIA	A	6	*	Clave única de la Gerencia
GERENCIA	A	45		Descripción de la Gerencia

GERENCIAS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
RFC	A	10	*	Registro Federal de causantes del responsable
NOMBRE	A	30		Nombre del responsable
TELEFONO	A	10		Teléfono del responsable
EXTENSION	A	5		Extensión del responsable

RESPONSABLES

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_SUBGERENCIA	A	6	*	Clave única de la Subgerencia
SUBGERENCIA	A	45		Descripción de la Subgerencia

SUBGERENCIAS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_TIPO	A	2	*	Clave única del tipo de proyecto
TIPO	A	45		Descripción del tipo de proyecto

TIPOS DE PROYECTO

TABLAS DE CAPTURA

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_PROYECTO	A	8	*	Clave única del proyecto
CVE_TIPO	A	2		Clave única del tipo del proyecto
CVE_ESTADOP	A	1		Clave única del estado del proyecto
CVE_GERENCIA	A	6		Clave única de la Gerencia
CVE_SUBGERENCIA	A	6		Clave única de la Subgerencia
CVE_EMPRESA	A	6		Clave única de la Empresa
RFC	A	10		RFC del responsable
FECHA_INICIO	D			Fecha de inicio del proyecto
FECHA_TERMINO	D			Fecha de término del proyecto
COSTO	N			Costo programado del proyecto
NOMBRE	A	150		Nombre del proyecto
RENGLON	A	3		Renglón del gasto del proyecto

PROYECTOS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_PROYECTO	A	8	*	Clave única del proyecto
AÑO	A	4	*	Año asignado para el presupuesto
CANTIDAD	N			Cantidad presupuestal

PRESUPUESTOS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_FACTURA	A	11	*	Clave única de la factura
CVE_PROYECTO	A	8	*	Clave única del proyecto
CVE_ESTADOF	A	1		Clave única del estado de la factura
FECHA	D			Fecha de elaboración de la factura
PERIODO	D			Período que ampara la factura
TOTAL	N			Cantidad erogada por la factura
CONCEPTO	A	200		Justificación de la factura

FACTURAS

NOMBRE DEL CAMPO	TIPO	TAM.	K	DESCRIPCION
CVE_PROYECTO	A	8	*	Clave única del proyecto
CVE_FACTURA	A	11	*	Clave única de la factura
FECHA_ENTRADA	D			Fecha de entrada de la factura a la Dependencia
FECHA_SALIDA	D			Fecha de salida de la factura de la Dependencia
CVE_DEPENDENCIA	A	6		Clave única de la Dependencia
BANDERA		1		Indica el último lugar donde se encuentra la factura

UBICACIONES DE LAS FACTURAS

Nota: K = (Llave), y los campos marcados con asterisco en esta columna muestran que son campos utilizados como llave única.

4.4.3 Manejo y descripción general del sistema

Para el buen funcionamiento y empleo de un sistema de software (en este caso el sistema de control de facturas y proyectos), es necesario tomar en cuenta el conocimiento del manejo de los menús, procesos (captura, consulta, opciones, teclas utilizadas, etc.). Para ello a continuación se da a conocer como funciona este sistema, sus módulos (captura, salidas, etc.) y la manera más eficaz de utilizarlo.

Instalación y ejecución

Para poder utilizar las capacidades del sistema de control de facturas y proyectos, es necesario instalarlo en un disco duro. A continuación se describe el proceso de instalación, así como la forma de ejecutar el sistema. Cabe mencionar que para instalar y por consiguiente ejecutar el sistema es necesario cumplir con ciertos requerimientos que a continuación se presentan.

Requerimientos de Software

1.-Windows versión 3.0 y posteriores.

2.-Manejador de Base de Datos **PARADOX FOR WINDOWS** versión 1.0.

Requerimientos de Hardware

1.-Computadora PC.

a) Procesador 386 SX ó DX y posteriores.

b) Velocidad de operación 33 MHz mínimo.

c) Memoria de RAM 4 MB mínimo (recomendable 8 MB).

d) Disco duro de 100 MB (recomendable).

2.-Impresora.

a) HP Laserjet III (recomendable).

INSTALACION

Antes de instalar el sistema "control de facturas y proyectos" es necesario tomar muy en cuenta los requerimientos antes mencionados, ya que en caso de no cumplir con alguno de ellos no se garantiza poder ni siquiera instalarlo.

Tomando en cuenta que Windows y PARADOX for Windows se encuentran instalados:

a) Se crea un subdirectorio en el directorio PDOXWIN llamado **CONFYP**

C:\CD PDOXWIN

C:\PDOXWIN\MD CONFYP

C:\PDOXWIN\CD CONFYP

C:\PDOXWIN\CONFYP

b) Se crea un ícono (como el mostrado en la figura 4.9) para ejecutar desde Windows el sistema

- Creando un grupo (CONFYP)
- Creando un objeto dentro del grupo (CONFYP)
- Considerando las propiedades siguientes:

Descripción: **CONFYP**

Línea de comando: **C:\PDOXWIN\PDOXWIN.EXE**
C:\PDOXWIN\CONFYP\GENERAL.SSL

Directorio de trabajo: **C:\PDOXWIN**

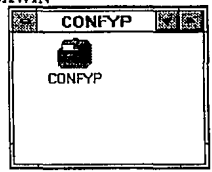


Figura 4.9 Icono del CONFYP

EJECUCION

Una vez realizado el proceso de instalación, ya se tiene la posibilidad de ejecutar el sistema. Para esto solamente se ejecuta el ícono del CONFYP.

Teclas de operación del sistema

A continuación en la figura 4.10 se describen las funciones de todas las teclas que se utilizan en el manejo del sistema.

TECLA	FUNCION
→	Mueve un campo a la derecha.
←	Mueve un campo a la izquierda.
↑	Desplaza un campo hacia arriba.
↓	Desplaza un campo hacia abajo.
PGUP	Retrocede a la pantalla anterior.
PGDOWN	Avanza a la pantalla siguiente.
[CTRL] HOME	Mueve el cursor al primer registro de la pantalla.
[CTRL] END	Mueve el cursor al último registro de la pantalla.
<Enter>	Acepta el valor dado para el campo sobre el que se encuentra el cursor ó avanza un campo.
INSERT	Inserta un registro
TAB	Mueve el cursor de campo en campo
[CTRL] DEL	Suprime el registro en el cual este en posición el cursor
[CTRL] (CARACTER)	Activa el menú y según el carácter oprimido se accesa a la opción correspondiente

Figura 4.10 Teclas de ayuda usadas durante la operación del sistema

4.4.4 Diagramas a bloques y de flujo del sistema

De acuerdo a los requerimientos del sistema, es necesario considerar un diagrama de bloques general (menú principal) del sistema el cual se presenta a continuación en la figura 4.11. Donde se presentan todas las opciones que contiene el sistema.

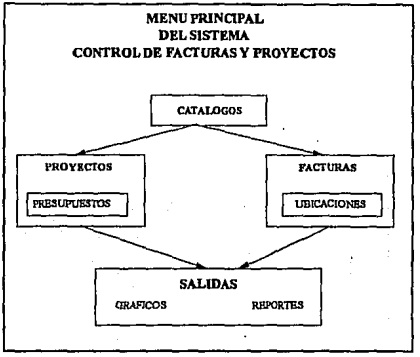


Figura 4.11 Diagrama a bloques general del sistema control de facturas y proyectos

4.4.4.1 Menú principal

El sistema de control de facturas y proyectos cuenta con un menú principal, donde se permite al usuario accesar a diferentes opciones, como son:

OPCIONES
Catálogos
Facturas
Proyectos
Reportes
Salida

El diagrama de flujo correspondiente al menú principal del sistema se presenta a continuación en la figura 4.12 donde el proceso que actúa entre el menú principal y cada una de las opciones es llamando a un proceso, en caso de elegir una opción.

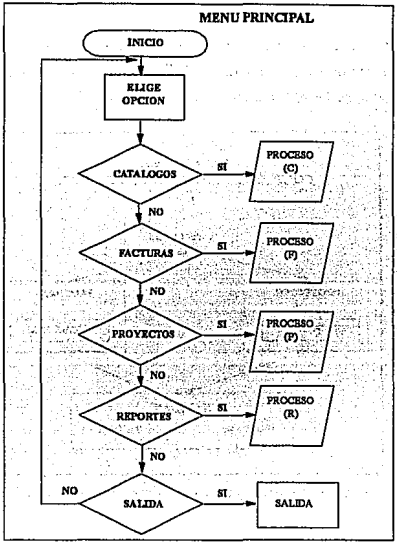


Figura 4.12 Diagrama de flujo general del sistema control de facturas y proyectos

La pantalla que se muestra a continuación en la figura 4.13 es la representación del diagrama general del sistema.

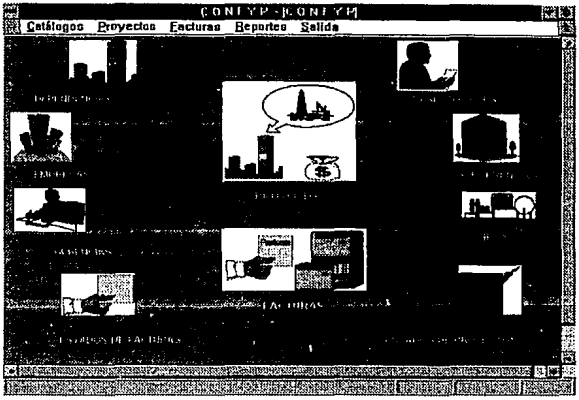


Figura 4.13 Menú principal del sistema

Los programas fuentes que contienen los métodos y procedimientos empleados en esta opción se presentan en el punto 2 (Menú principal) del Apéndice B (Programas fuentes).

Como se muestra en la figura 4.13 las opciones con que cuenta el menú principal del sistema son cinco y a continuación se ofrece una pequeña explicación de cada una de éstas.

Catálogos: son datos en forma de lista que contribuyen a formar la base de datos del sistema, por ejemplo, proyectos, facturas e informes. Esta opción permite observar, cancelar y/o modificar los catálogos con los que cuenta el sistema.

Facturas: es un documento que ampara una cantidad con respecto al avance de un proyecto. Al utilizar esta opción se permite dar de alta nuevas facturas, cancelar o modificar las ya existentes.

Proyectos: es un conjunto de actividades que se realizan con la finalidad de lograr un objetivo específico. Al llegar a este punto el usuario tiene la posibilidad de dar de alta proyectos nuevos así como cancelar o modificar los ya existentes.

Reportes: es un documento que nos proporciona información de los datos del sistema. Con la opción "Reportes", el sistema genera los reportes correspondientes al control de proyectos y de sus facturas.

El acceso a una opción del menú o a un ícono se puede realizar mediante las teclas de función que se encuentran en la figura 4.10. El menú y los íconos correspondientes realizan las mismas acciones, (ver figura 4.14).

CATALOGO	DESCRIPCION	UTILIDAD
Dependencias	Incluye todas las Dependencias de Pemex relacionadas con el proceso de Autorización de la Factura.	Este catálogo es accesado desde el módulo de Ubicaciones de las facturas contenido en la opción facturas.
Empresas	Incluye todas las Empresas que realizan proyectos para Pemex.	Este catálogo es accesado desde el módulo de proyectos.
Gerencias	Incluye todas las Gerencias de Pemex que tienen a su cargo proyectos a desarrollar.	Este catálogo es accesado desde el módulo de proyectos.
Estados de Factura	Es el estado de la factura y puede ser: [A,D,R,T].	Este catálogo es accesado desde el módulo de facturas.
Estados de Proyecto	Es el estado del proyecto y puede ser: [A]utorizado [C]ancelado [S]uspendido [T]erminado	Este catálogo es accesado desde el módulo de proyectos.
Tipos de Proyecto	Es una clasificación del proyecto.	Este catálogo es accesado desde el módulo de proyectos.
Subgerencias	Incluye todas las Subgerencias de Pemex que tienen a su cargo el desarrollo de proyectos.	Este catálogo es accesado desde el módulo de proyectos.
Responsables	Son los responsables por parte de Pemex de los proyectos a desarrollar.	Este catálogo es accesado desde el módulo de proyectos.

Figura 4.14 Descripción de los catálogos de sistema control de facturas y proyectos

4.4.4.2 Catálogos

Dentro del menú principal del sistema la opción catálogos se encuentra activa. En caso de elegir esta opción se despliega un submenú con todos (8) los catálogos que contiene el sistema, permitiendo al usuario elegir uno de ellos para accesarlo.

El diagrama a bloques se presenta en la figura 4.15 y el diagrama de flujo correspondiente en la figura 4.16.

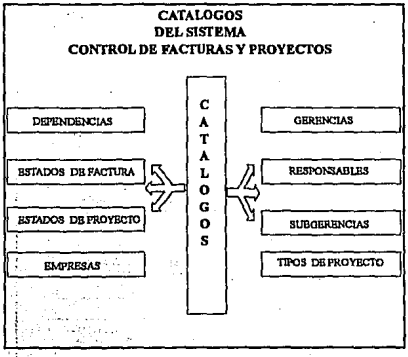


Figura 4.15 Diagrama a bloques de catálogos

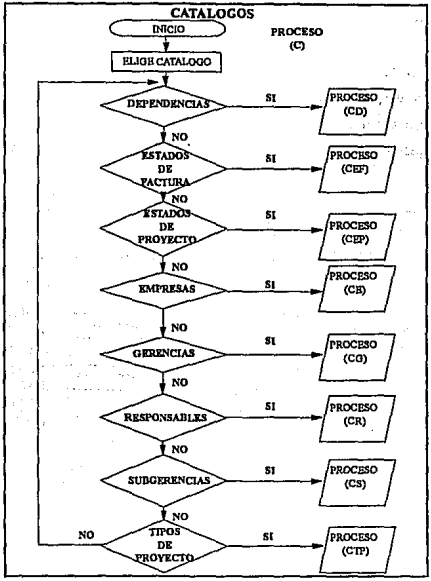


Figura 4.16 Diagrama de flujo de catálogos

Todos los catálogos siguen una misma metodología para realizar la captura (altas, bajas y modificaciones) y consultas de sus datos, a continuación se presenta en la figura 4.17 el diagrama a bloques representativo del catálogo de Empresas indicando los procesos que puede realizar estando en esta opción. (Ver Apéndice B punto 3 catálogo de Empresas).

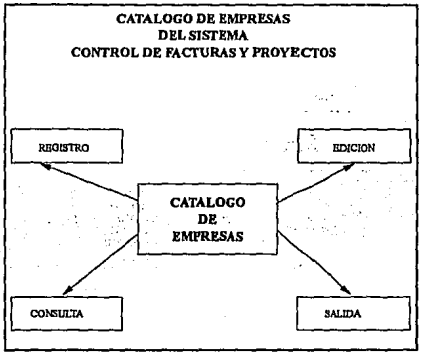


Figura 4.17 Diagrama a bloques del catálogo Empresas

El diagrama de flujo del catálogo de Empresas se presenta a continuación en la figura 4.18.

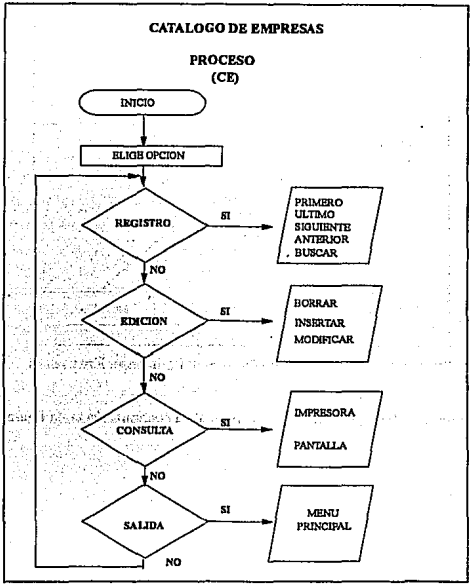


Figura 4.18 Diagrama de flujo del catálogo de Empresas

En el catálogo Empresas como en todos los demás del sistema, se maneja una sola pantalla para realizar todos los procesos de captura (altas,bajas, modificaciones y el reporte del catálogo). La figura 4.19 que a continuación se muestra, es la pantalla del catálogo de Empresas donde se visualiza un menú en la parte superior de esta, relacionado con los íconos que encuentran un poco más abajo del menú en la parte izquierda; indicando con ello que el acceso a un proceso deseado se podrá realizar ya sea eligiendo una opción del menú u oprimiendo el ícono correspondiente. Las teclas que ayudan al manejo de este catálogo se encuentran en la figura 4.10.

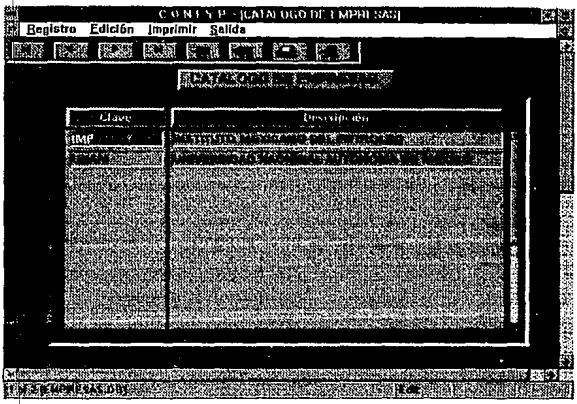


Figura 4.19 Catálogo de Empresas

Los procesos que se utilizan en los catálogos son:

a) "Registro"

Primero	-Posiciona el cursor en el primer registro de la tabla.
Ultimo	-Posiciona el cursor en el último registro de la tabla.
Siguiente	-Posiciona el cursor en el siguiente registro apartir del cual este seleccionado.
Anterior	-Posiciona el cursor en el anterior registro apartir del cual este seleccionado.
Buscar	-Busca un campo deseado en la tabla correspondiente.

b) "Edición"

Borrar	-Borra el registro en posición de la lista.
Insertar	-Inserta un nuevo registro en la tabla.
Modificar	-Modifica el registro en posición de la tabla.

c) "Imprimir" -Procede a obtener un reporte con todos los registros contenidos en el catálogo.

d) "Salida"

Si -Procede a salir de la opción de catálogo.
No -Cancela la opción seleccionada.

Nota: Todos los catálogos de este sistema tienen un manejo semejante al mencionado anteriormente por ello que en este punto no se hace referencia a ellos y se toma como base el catálogo Empresas.

TABLA	CAMPO	FORMATO DE CAPTURA
Empresas	Clave	[6] Caracteres alfanuméricos
Empresa	Descripción	[45] Caracteres alfanuméricos
Estados de Factura	Clave	[1] Caracteres alfanuméricos
Estados de Factura	Descripción	[10] Caracteres alfanuméricos
Estados de Proyecto	Clave	[1] Caracteres alfanuméricos
Estados de Proyecto	Descripción	[10] Caracteres alfanuméricos
Tipos de Proyecto	Clave	[2] Caracteres alfanuméricos
Tipos de Proyecto	Descripción	[45] Caracteres alfanuméricos
Responsables	RFC	[10] Caracteres alfanuméricos
Responsables	Nombre	[30] Caracteres alfanuméricos
Responsables	Teléfono	[10] Caracteres alfanuméricos, 9-99-99-99
Responsables	Extensión	[5] Caracteres alfanuméricos

Figura 4.20 Formatos de captura de los campos de los catálogos

4.4.4.3 Proyectos

En la opción de proyectos se realizan los procesos de captura (altas, bajas, modificaciones) y consultas (pantalla e impresora), como se muestran a continuación en la figura 4.21. En esta opción del menú principal se controla toda la información (técnica, económica y administrativa) de proyectos a desarrollar.

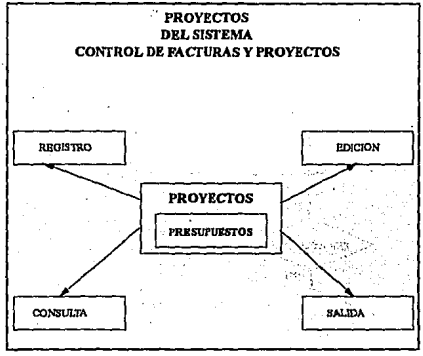


Figura 4.21 Diagrama a bloques de proyectos

El menú principal de la opción proyectos consta de ***REGISTRO, EDICION, CONSULTA Y SALIDA***; al igual que en todos los catálogos del sistema.
El diagrama de flujo utilizado en proyectos se presenta en la figura 4.22.

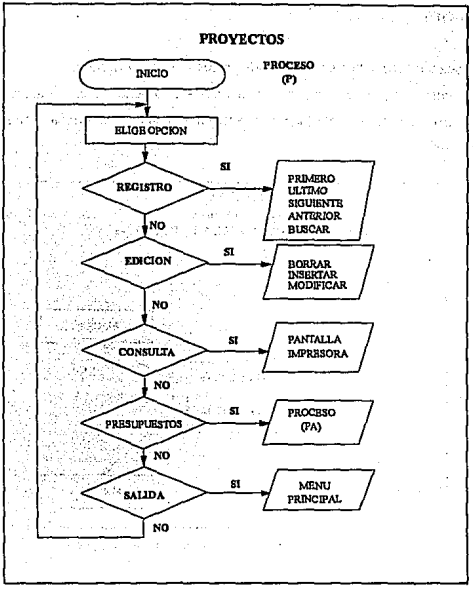


Figura 4.22 Diagrama de flujo de la opción proyectos

La opción de proyectos contiene un submódulo (presupuesto asignado), donde los procesos que tiene activados son los mismos que los de su módulo principal (proyectos). (Ver figura 4.23). Este pequeño módulo se encuentra incluido en proyectos debido a que la información presupuestal tiene una relación muchos a uno con proyectos, es decir, pueden existir varios presupuestos (anuales) para un proyecto, dependiendo de la duración del proyecto.

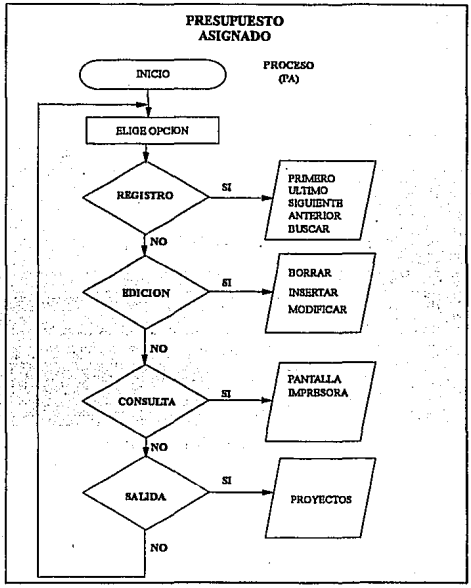


Figura 4.23 Diagrama de flujo de la opción proyectos (presupuesto asignado)

Al elegir la opción de proyectos se entra a una pantalla donde se visualiza el primer registro de la tabla proyectos, en caso de tenerlo. En esta pantalla se tienen las mismas opciones que se consideran para todos los catálogos, y existe también el mismo menú en la parte superior de la pantalla y sus íconos correspondientes como se observa a continuación en la figura 4.24.

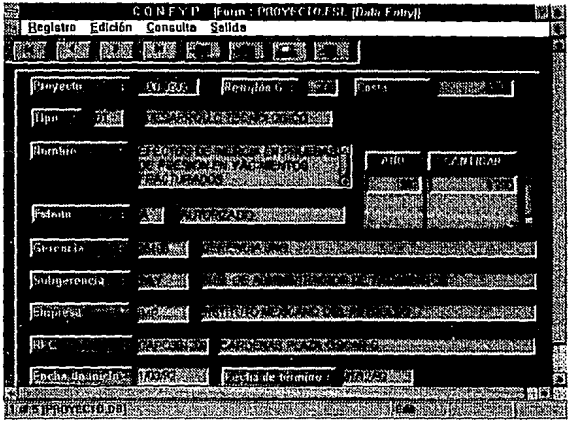


Figura 4.24 Captura de proyectos

Los campos que se utilizan en la opción de proyectos se presentan a continuación en la figura 4.25, donde se presenta su formato de captura; en caso de que por algún motivo no se dé correctamente el formato de un campo, el sistema sensa el error de formato del campo y, manda un mensaje indicando al usuario el error cometido; para que se corrija en el momento. El sistema no permite la captura de información no válida, por ello es que el sistema manda mensajes de error y hasta no capturar información correcta se continua con el proceso de captura.

Es necesario indicar que existen campos donde se emplea un tipo de catálogo. Es decir en caso de tener la clave del campo requerido se hace uso del catálogo en cuestión. En los formatos de captura de campos se inicia en algunos con un asterisco, mostrando con ello que este campo requiere de un catálogo.

CAMPO	FORMATO DE CAPTURA
Proyecto	[8] Caracteres alfanuméricos
Renglón	Numérico [3] Enteros, [208-Inv. o 308-Oper.]
Costo	Numérico
*Tipo	[2] Caracteres alfanuméricos
Nombre	[150] Caracteres alfanuméricos
*Estado	[1] Carácter alfanumérico
*Gerencia	[6] Caracteres alfanuméricos
*Subgerencia	[6] Caracteres alfanuméricos
*Empresa	[6] Caracteres alfanuméricos
*RFC	[10] Caracteres alfanuméricos
Fecha de inicio	Fecha [DD/MM/YY]
Fecha de término	Fecha [DD/MM/YY]
Año	Numérico [4] Enteros
Cantidad	Numérico

Figura 4.25 Formatos de captura de los campos de proyectos

En el menú de proyectos presentado en la figura 4.24 se encuentra la opción de **"Consulta"** a diferencia del menú de catálogos.
La opción **"Consulta"** accesa los procesos de consulta por pantalla o generación del reporte por impresora.

"Consulta"

Impresora -Genera el reporte por impresora, el cual contiene todos los proyectos dados de alta.

Pantalla -Genera el reporte por pantalla, el cual contiene todos los proyectos dados de alta. Una consulta de proyectos por pantalla se muestra a continuación en la figura 4.26.

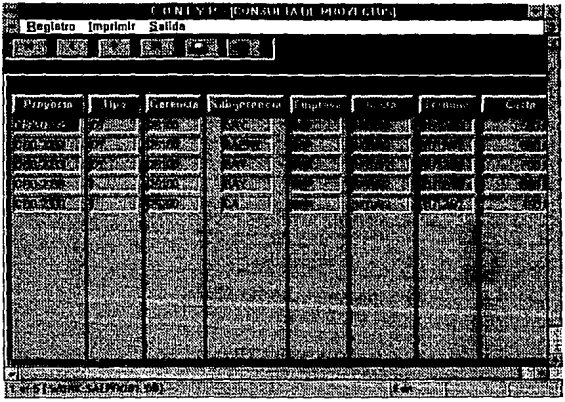


Figura 4.26 Consulta de proyectos

Todos los programas fuentes de esta opción de proyectos se encuentran en el punto 4 (Proyectos) del Apéndice B (Programas Fuentes del Control de Facturas y Proyectos). En este Apéndice se muestran todos los procedimientos, métodos, querys y nombres de formas y tablas que esta opción utiliza.

4.4.4.4 Facturas

En esta opción de facturas se permite al usuario capturar toda la información relacionada con las facturas erogadas por los proyectos existentes
La razón de que exista una factura tiene que ver totalmente con la existencia de un proyecto por el cual la factura es creada. Pueden existir varias facturas que tengan relacionadas un proyecto. Existe entonces una relación un proyecto a muchas facturas.

A continuación se muestran los diagramas a bloques y de flujo de la opción de facturas del sistema de control de facturas y proyectos, en las figuras 4.27, 4.28 y 4.29. Donde se describe de manera ejemplificada las opciones.

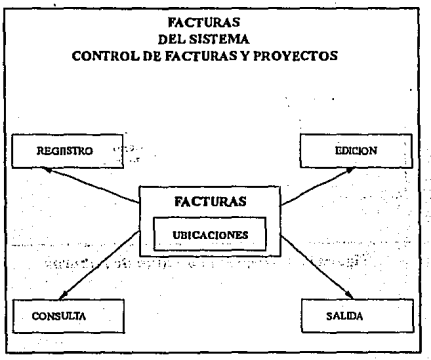


Figura 4.27 Diagrama a bloques de facturas

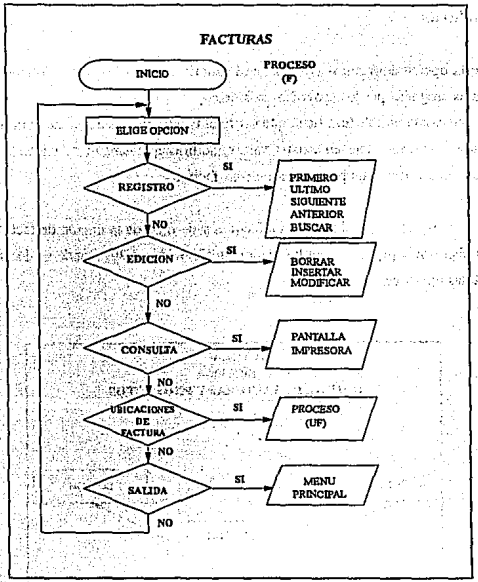


Figura 4.28 Diagrama de flujo de facturas

Dentro del módulo de facturas existe un submódulo de ubicaciones de facturas que contiene toda la información del proceso de facturación, desde su llegada a Pemex y su correspondiente aceptación (el paso de la factura a todas las Dependencias que deben dar el visto bueno y su consiguiente aceptación). Su diagrama de flujo se presenta a continuación en la figura 4.29.

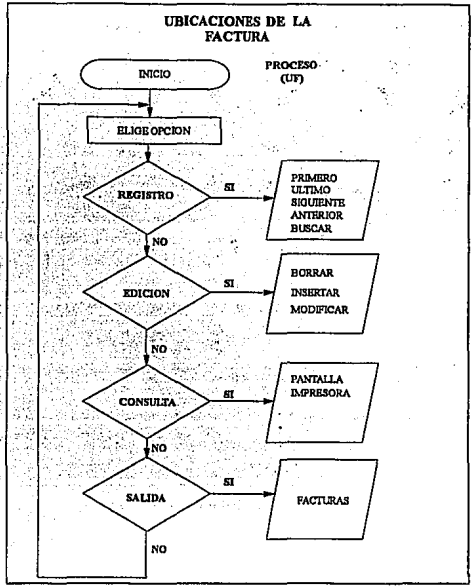


Figura 4.29 Diagrama de flujo de facturas (ubicaciones de la factura)

Al elegir la opción de facturas se entra a una primera pantalla como se muestra en la figura 4.30, donde se visualiza el primer registro de la tabla de facturas y quedando activas todas las opciones del menú y sus íconos correspondientes; en caso de no existir registros en la tabla de facturas la pantalla aparece con la opción de captura de facturas.

El manejo del menú se efectúa de manera semejante como en el módulo de proyectos y las opciones activadas en facturas son las mismas que en proyectos (registro, edición, consulta etc.)

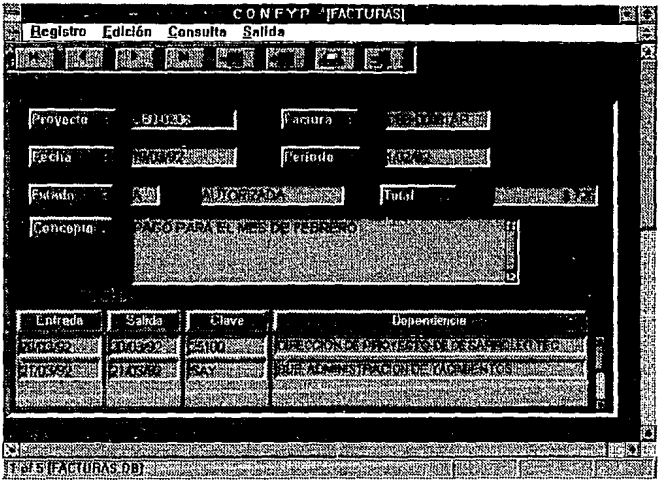


Figura 4.30 Captura de facturas

En el módulo facturas se considera también como información a capturar, los datos correspondientes a las fechas de entrada y salida de las facturas en las Dependencias de Pemex. Los campos a capturar en el módulo facturas se encuentran en la pantalla anterior. Los campos utilizados son los que aparecen en la figura 4.31.

CAMPO	FORMATO DE CAPTURA
Factura	[11] Caracteres alfanuméricos
Proyecto	[8] Caracteres alfanuméricos
Fecha	Fecha [DD/MM/YY]
Período	Fecha [DD/MM/YY]
*Estado	[2] Caracteres alfanuméricos
Total	Numérico
Concepto	[150] Caracteres alfanuméricos
Fecha de entrada	Fecha [DD/MM/YY]
Fecha de salida	Fecha [DD/MM/YY]
*Dependencia	[6] Caracteres alfanuméricos

Figura 4.31 Formatos de captura de los campos de facturas

En el módulo facturas se considera también como información a capturar, los datos correspondientes a las fechas de entrada y salida de las facturas en las Dependencias de Pemex.
Los campos a capturar en el módulo facturas son los mostrados en la pantalla anterior.
Una consulta de las facturas del sistema por pantalla se encuentra enseguida en la figura 4.32 y el reporte se puede obtener también por impresora.

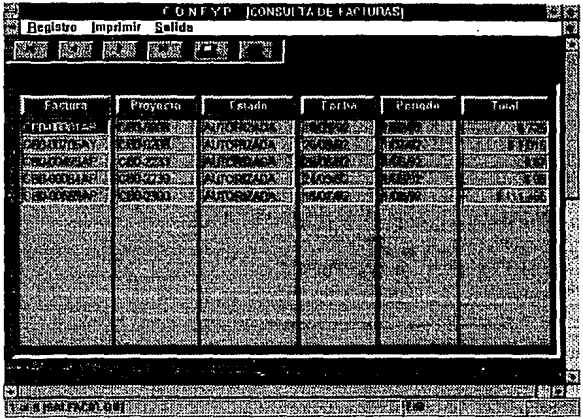


Figura 4.32 Consulta de facturas

Todos los programas fuentes de esta opción de proyectos se encuentran en el punto 5 (Proyectos) del Apéndice B (Programas Fuentes del Control de Facturas y Proyectos). En este Apéndice se describen todos los programas fuentes, es decir todos los métodos y procedimientos utilizados para generar los procesos que utiliza esta opción de proyectos.

4.4.4.5 Reportes

Al elegir esta opción se accesa al módulo de reportes el cual contiene todas las salidas que genera el sistema de control de facturas y proyectos, a continuación se presentan los diagramas a bloques y de flujo del módulo de reportes. Los programas fuentes que generan todas los procesos y opciones de reportes se muestran en el Apéndice B (Programas Fuentes) en el punto 6 (Reportes). Para generar cada uno de los reportes es necesario la generación de una tabla para que apartir de ahí se obtengan los reportes. La creación de la tabla, se realiza por medio de un query. Por lo tanto cada reporte tiene relacionado un query y una tabla.

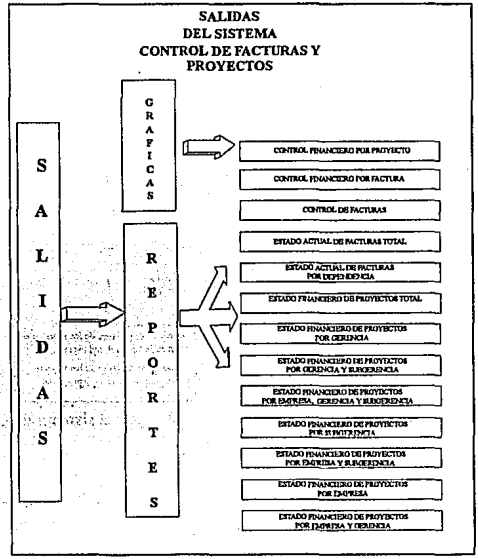


Figura 4.33 Diagrama a bloques de las salidas obtenidas por el sistema de Control de Facturas y Proyectos

En la figura 4.33 se presenta el diagrama a bloques de los reportes que genera el sistema, los cuales se obtienen por pantalla y por impresora.

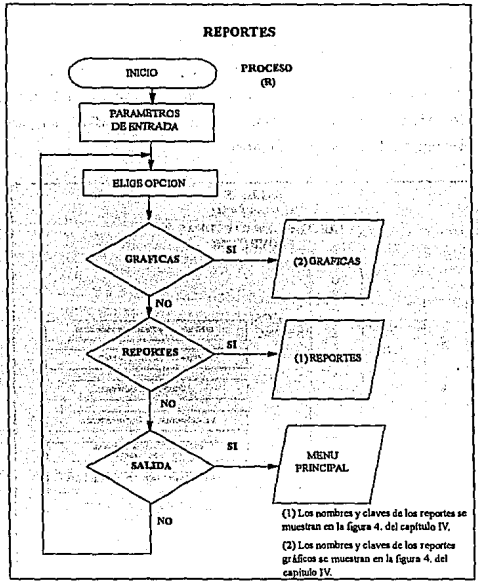


Figura 4.34 Diagrama de flujo de las salidas obtenidas por el sistema de Control de Facturas y Proyectos

Para la generación de los reportes se utiliza el diagrama de flujo de la fig. 4.34, una vez que se ingresan los datos se selecciona el reporte, el cual se obtiene por pantalla e impresora.

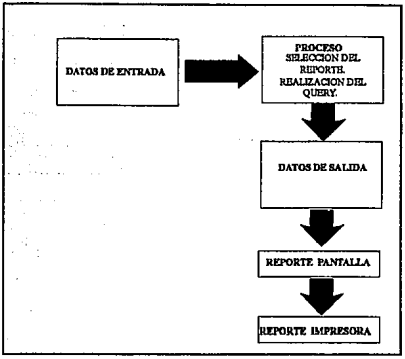


Figura 4.35 Diagrama a bloques general del proceso de generación de las salidas

En la figura 4.35 se muestra el diagrama general a bloques que se utiliza para obtener cualquiera de los reportes que genera el sistema, como se observa en el diagrama, primero se ingresan los datos de entrada, estos parámetros se toman en cuenta para generar un **Query** (relaciones entre tablas), con el cual se obtiene una tabla que contendrá los datos del reporte. El reporte se obtiene por default por pantalla, dejando la opción de obtenerlo por impresora si así se desea.

Las salidas de cualquier sistema de software, son quizá lo más importe del sistema ya que de ello depende en gran parte la justificación del sistema y su total funcionalidad. Los reportes que se generan en el sistema de control de facturas y proyectos son:

No.	NOMBRE DEL REPORTE
REPORTE G	CONTROL FINANCIERO POR PROYECTO

Figura 4.36 Gráficas

No.	NOMBRE DEL REPORTE
REPORTE 1	CONTROL FINANCIERO POR PROYECTO
REPORTE 2	CONTROL FINANCIERO POR FACTURA
REPORTE 3	CONTROL DE FACTURAS
REPORTE 4A	ESTADO ACTUAL DE FACTURAS TOTAL
REPORTE 4B	ESTADO ACTUAL DE FACTURAS POR DEPENDENCIA
REPORTE 5A	ESTADO FINANCIERO DE PROYECTOS TOTAL
REPORTE 5B	ESTADO FINANCIERO DE PROYECTOS POR GERENCIA
REPORTE 5C	ESTADO FINANCIERO DE PROYECTOS POR GERENCIA Y SUBGERENCIA
REPORTE 5D	ESTADO FINANCIERO DE PROYECTOS POR EMPRESA,GERENCIA SUBGERENCIA
REPORTE 5E	ESTADO FINANCIERO DE PROYECTOS POR SUBGERENCIA
REPORTE 5F	ESTADO FINANCIERO DE PROYECTOS POR EMPRESA Y SUBGERENCIA
REPORTE 5G	ESTADO FINANCIERO DE PROYECTOS POR EMPRESA
REPORTE 5H	ESTADO FINANCIERO DE PROYECTOS POR EMPRESA Y GERENCIA

Figura 4.37 Reportes del sistema control de facturas y proyectos

La descripción de los reportes de la figura 4.37 es la siguiente :

Control financiero de proyectos de la Gerencia de Desarrollo Tecnológico (GDT)

Este tipo de reporte proporciona toda la información correspondiente al proyecto especificado, presupuesto etc. El reporte se genera tanto por pantalla como por impresora, además de tener el reporte gráfico.

Control financiero por factura

Este reporte obtiene toda la información relacionada con una factura. Este reporte se genera tanto por pantalla como por impresora.

Control de facturas

Este reporte proporciona todas las facturas capturadas en un período de tiempo. Este reporte se genera tanto por pantalla como por impresora.

Estado actual de facturas

El estado actual de facturas reporta las ubicación de las facturas en las Dependencias de Pemex correspondientes. Este reporte se puede generar por las facturas que se encuentran en una

Dependencia o en todas las dependencias; considerando un período de tiempo. Este reporte se genera tanto por pantalla como por impresora.

Estado financiero de proyectos

Este reporte proporciona la información más relevante de los proyectos considerando un período de tiempo. El estado financiero de proyectos puede obtenerse por:

a) Empresa:	**Este reporte se obtiene tanto en pantalla como por impresora.**
b) Gerencia:	**Este reporte se obtiene tanto en pantalla como por impresora.**
c) Subgerencia:	**Este reporte se obtiene tanto en pantalla como por impresora.**
d) Gerencia y Subgerencia:	**Este reporte sólo se genera por pantalla.**
e) Gerencia y Empresa:	**Este reporte sólo se genera por pantalla.**
f) Empresa y Subgerencia:	**Este reporte sólo se genera por pantalla.**
g) Empresa, Gerencia y Subgerencia:	**Este reporte sólo se genera por pantalla.**
h) Total:	**Este reporte sólo se genera por pantalla.**

Para la generación de todos los reportes es necesario considerar ciertos parámetros de entrada para obtener el reporte deseado. Dependiendo del reporte se consideran los parámetros, aunque existen algunos que son generales y necesarios para todos los reportes como son:

Fecha inicial: donde el formato de captura es [DD/MM/YY]

Fecha final: donde el formato de captura es [DD/MM/YY]

Y donde la fecha final debe ser mayor que la fecha inicial dada.

La pantalla que se muestra a continuación en la figura 4.38 contiene todos los datos que se pueden requerir para obtener los reportes, y los datos se ingresan de acuerdo al reporte que se quiera generar.

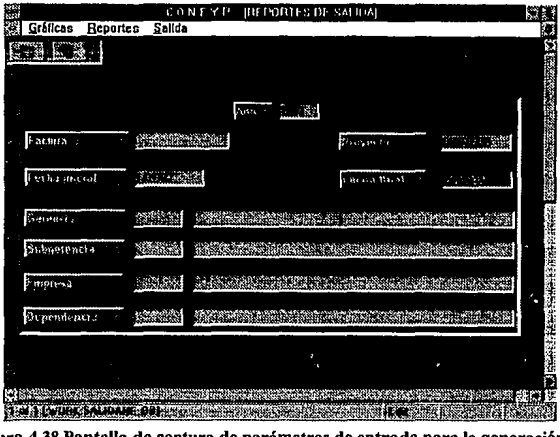


Figura 4.38 Pantalla de captura de parámetros de entrada para la generación de reportes del sistema control de facturas y proyectos

A continuación se presenta la descripción particular de los diagramas a bloque y flujo para cada uno de los reportes que genera el sistema, como se observa en las siguientes figuras, los diagramas son similares, lo único que varía en los diagramas son los datos de entrada y el nombre del **Query** (relaciones entre tablas) que se utiliza.

Control Financiero de los Proyectos

El diagrama general a bloques del reporte control financiero de proyectos se muestra en la figura 4.39. Este reporte en particular se obtiene también en forma gráfica. En todos los diagramas se utiliza la notación (*Rn), donde n es un indicador del número de reporte, para especificar cuales son los datos de entrada, los cuales varían para cada reporte, así mismo se indica en cada diagrama el nombre del **Query** (relaciones entre tablas) que se utiliza para generar cada reporte. El **Query** relaciona los datos de la tabla **SALIDARE.DB** con otras tablas, dando como resultado para este reporte la tabla (**SALCON01.DB**) (Ver apéndice A).

Todos los reportes en el proceso del query generan una tabla para cada uno de ellos cuyo nombre es igual al del query, con la extensión (**.DB**).

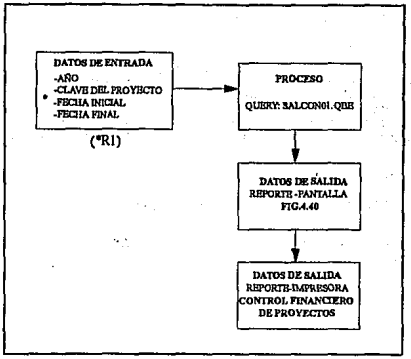


Figura 4.39 Diagrama a bloques para generar el reporte control financiero de proyectos

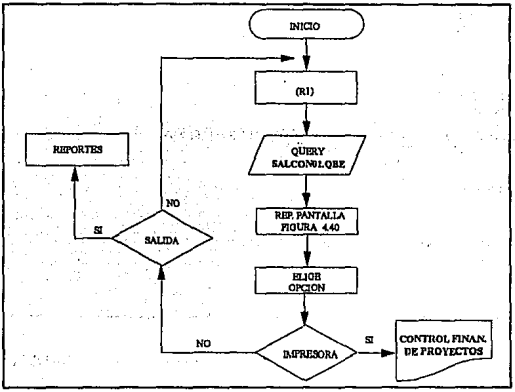


Figura 4.40 Diagrama de flujo del reporte control financiero de proyectos

La figura 4.40 representa el diagrama de flujo para el reporte control financiero de proyectos. El query utilizado es el siguiente:

```
QUERY PARA GENERAR LA TABLA DE SALIDA PARA EL REPORTE 1
Query
ANSWER: :WORK:SALCON01.DB

SORT: PROYECTO.DB->"CVE_PROYECTO", FACTURAS.DB->"CVE_FACTURA",
FACTURAS.DB->"PERIODO", SALIDARE.DB->"AÑO", PROYECTO.DB->"FECHA_INICIO",
PROYECTO.DB->"FECHA_TERMINO", PROYECTO.DB->"NOMBRE", TIPOS.DB->"TIPO",
EDOPROY.DB->"EDO_PROYECTO", GERENCIA.DB->"GERENCIA",
SUBGEREN.DB->"SUBGERENCIA", EMPRESAS.DB->"EMPRESA", RESPONSA.DB->"NOMBRE",
FACTURAS.DB->"TOTAL", EDOFACT.DB->"EDO_FACTURA", PRESUPUE.DB->"CANTIDAD",

SALIDARE.DB|AÑO      |CVE_PROYECTO|FECHA_INICIAL|FECHA_FINAL|
      |Check _EG08|_EG01     |_EG10     |_EG11     |

PROYECTO.DB|CVE_PROYECTO|CVE_TIPO|CVE_ESTADOP|CVE_GERENCIA|
      |Check _EG01 |_EG02  |_EG03    |_EG04     |

PROYECTO.DB|CVE_SUBGERENCIA|CVE_EMPRESA|RFC  |FECHA_INICIO|
      |_EG05      |_EG06    |_EG07|Check     |

PROYECTO.DB|FECHA_TERMINO|NOMBRE|
      |Check     |Check |

TIPOS.DB|CVE_TIPO|TIPO  |
     |_EG02  |Check |

EDOPROY.DB|CVE_ESTADOP|EDO_PROYECTO|
      |_EG03    |Check     |

GERENCIA.DB|CVE_GERENCIA|GERENCIA|
      |_EG04    |Check  |

SUBGEREN.DB|CVE_SUBERENCIA|SUBGERENCIA|
      |_EG05      |Check     |

EMPRESAS.DB|CVE_EMPRESA|EMPRESA|
      |_EG06    |Check  |

RESPONSA.DB|RFC  |NOMBRE|
      |_EG07|Check |

FACTURAS.DB|CVE_FACTURA|CVE_PROYECTO|CVE_ESTADOF|PERIODO     |
      |Check     |_EG01     |_EG09     |Check >=_EG10|
      |          |          |          |<=_EG11     |

FACTURAS.DB|TOTAL |
      |Check |
      |    |

EDOFACT.DB|CVE_ESTADOF|EDO_FACTURA|
      |_EG09    |Check     |

PRESUPUE.DB|CVE_PROYECTO|AÑO  |CANTIDAD|
      |_EG01     |_EG08|Check  |
EndQuery
```

El reporte por pantalla es el de la figura 4.41 y el de impresora se presenta al final del capítulo.

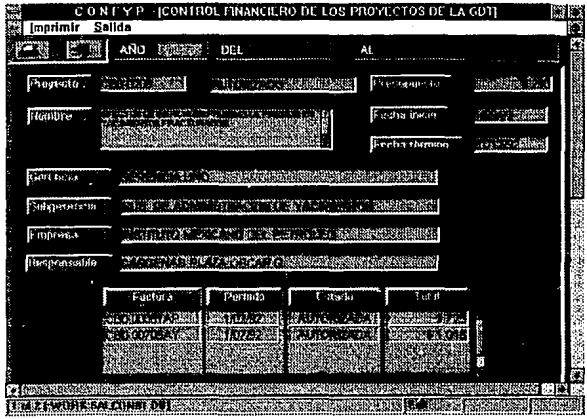


Figura 4.41 Reporte de control financiero por proyecto

El reporte de control financiero por proyecto que se obtiene, es muy utilizado en la Gerencia de Desarrollo Tecnológico, como se puede observar en la figura 4.41 el proyecto que se ingresó es el CBO-0206 (clave que determina la Gerencia), por medio de esta clave el sistema proporciona al usuario todas las facturas con que cuenta un proyecto y el estado de las mismas.

Este reporte proporciona a la Gerencia de Desarrollo Tecnológico el estado de un proyecto, en cuanto a los pagos que se han efectuado al mismo, y con estos datos controlan lo que se debe pagar para las próximas facturas sin exceder el presupuesto con que cuenta el proyecto, o en su defecto incrementar el presupuesto asignado. Todo dependerá de lo que acuerde la Gerencia de Desarrollo Tecnológico con la compañía que realiza el proyecto, pues en muchas ocasiones un proyecto es cancelado cuando excede su costo inicial.

Este tipo de reportes es de mucha utilidad debido a que a pesar de desplegar solamente la información de un sólo proyecto, puede mantenerse el control financiero (lo erogado) del proyecto además de tener conocimiento de (Nombre, Responsable de proyecto, Gerencia, Subgerencia, Empresa, Tipo, Presupuestos, Fecha de inicio, Fecha de término y Estado)

En la parte inferior del reporte (figura 4.41) se observan todas las facturas relacionadas a el proyecto, teniendo con ello el control financiero del proyecto. Lo único que se muestra en el reporte de las facturas es la clave de la factura, el periodo, el estado de la factura y el monto.

Información de proyectos:

PROYECTO
Clave
Nombre
Estado
Gerencia
Subgerencia
Empresa
Responsable
Presupuesto anual
Fecha de inicio
Fecha de término

Información de facturas:

FACTURA
Clave
Periodo
Estado
Total

Control Financiero por Factura

El siguiente reporte que se presenta es el de control financiero por factura al igual que el reporte anterior una vez que se seleccionan los datos de entrada se escoge el reporte que se desea generar, y los diagramas correspondientes se presentan a continuación en la figura 4.42.

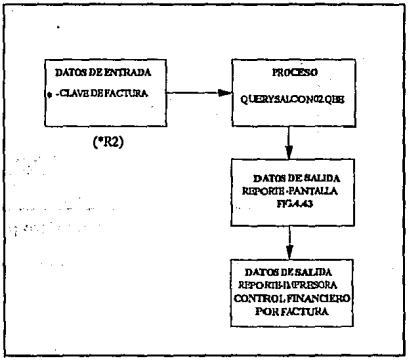


Figura 4.42 Diagrama a bloques del reporte control financiero por factura

El diagrama de flujo correspondiente a la figura 4.42 para generar el reporte control financiero por factura se muestra a continuación en la figura 4.43. El reporte se obtiene por pantalla y se encuentra en la figura 4.44 donde se observan los datos principales de la factura y las Dependencias donde ha estado.

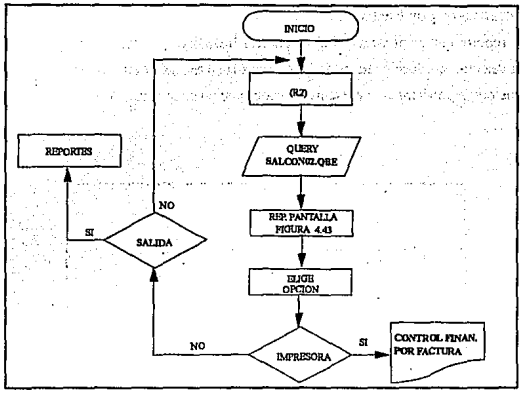


Figura 4.43 Diagrama de flujo del reporte control financiero por factura

```
QUERY PARA GENERAR LA TABLA DE SALIDA PARA EL REPORTE 2
Query

ANSWER: :WORK:SALCON02.DB

SORT: FECHFACT.DB->"FECHA_ENTRADA", FECHFACT.DB->"FECHA_SALIDA",
FACTURAS.DB->"CVE_FACTURA", FACTURAS.DB->"CVE_PROYECTO", FACTURAS.DB->"FECHA",
FACTURAS.DB->"PERIODO", FACTURAS.DB->"TOTAL", FACTURAS.DB->"CONCEPTO",
EDOFACT.DB->"EDO_FACTURA", DEPENDEN.DB->"DEPENDENCIA",

 SALIDARE.DB | CVE_FACTURA |
       | _EG01     |

 FACTURAS.DB | CVE_FACTURA | CVE_PROYECTO | CVE_ESTADOF | FECHA | PERIODO |
       | Check _EG01 | Check      | _EG02      | Check | Check  |

 FACTURAS.DB | TOTAL  | CONCEPTO |
       | Check | Check  |

 EDOFACT.DB | CVE_ESTADOF | EDO_FACTURA |
       | _EG02     | Check     |

 FECHFACT.DB | CVE_FACTURA | FECHA_ENTRADA | FECHA_SALIDA | CVE_DEPENDENCIA |
       | _EG01     | Check      | Check      | _EG03      |

 DEPENDEN.DB | CVE_DEPENDENCIA | DEPENDENCIA |
       | _EG03      | Check     |
EndQuery
```

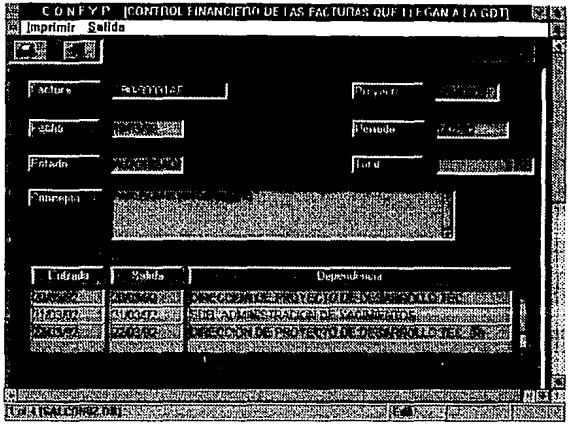


Figura 4.44 Reporte de control financiero por factura

La ventaja de obtener este reporte por pantalla e impresora, es el control que se tiene sobre las facturas, porque además de presentar todos los datos de una factura, facilita su localización en las diferentes Dependencias de Pemex. En particular este reporte es el más solicitado por la Gerencia de Desarrollo Tecnológico debido a que con frecuencia se desea conocer cuando entra y sale una factura de las Dependencias, pues en muchas ocasiones la factura tarda varios días en ser enviada a otra Dependencia y esto retrasa la llegada de la factura a la Dependencia llamada Gerencia de Desarrollo Tecnológico, el cual es el lugar a donde debe de llegar finalmente la factura, para que se apruebe el pago que se ha de realizar. Como se observa, el reporte genera la salida de los datos más importantes de una factura y el proyecto al que pertenece, pero sobretodo la trayectoria y el tiempo que tardo en cada una de las Dependencias hasta ser aprobada y enviada a la Gerencia Desarrollo Tecnológico.

Control de Facturas

El diagrama a bloques del reporte control de facturas se presenta en la figura 4.45 donde se observa al igual que en los anteriores diagramas los datos de entrada.

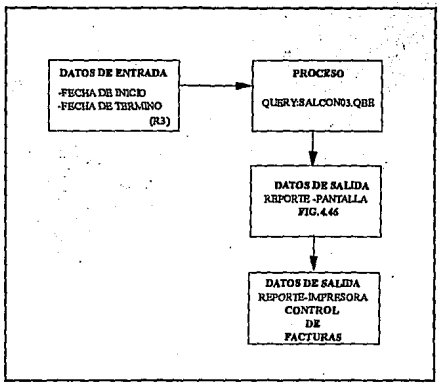


Figura 4.45 Diagrama a bloques del reporte control de facturas

Una vez obtenido el diagrama a bloques se obtiene el diagrama de flujo, en la figura 4.46.

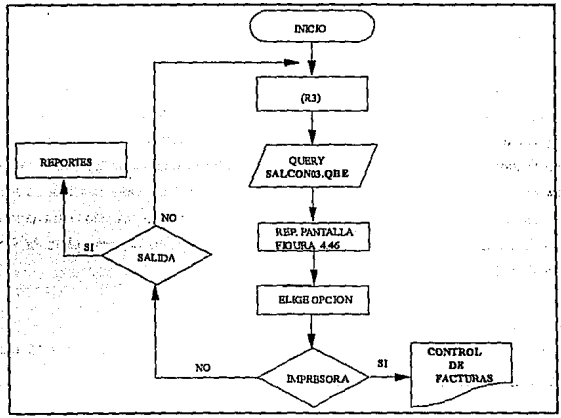


Figura 4.46 Diagrama de flujo del reporte control de facturas

```
QUERY PARA GENERAR LA TABLA DE SALIDA PARA EL REPORTE3
Query

ANSWER: :WORK:SALCON03.DB

SORT: FACTURAS.DB->"CVE_PROYECTO", FACTURAS.DB->"CVE_FACTURA",
FACTURAS.DB->"PERIODO", SALIDARE.DB->"FECHA_INICIAL",
SALIDARE.DB->"FECHA_FINAL", FACTURAS.DB->"FECHA", FACTURAS.DB->"TOTAL",
EDOFACT.DB->"EDO_FACTURA",

SALIDARE.DB | FECHA_INICIAL | FECHA_FINAL |
      | Check _EG04  | Check _EG05 |

FACTURAS.DB | CVE_FACTURA | CVE_PROYECTO | CVE_ESTADOF | FECHA |
      | Check _EG02 | Check _EG01  | _EG03    | Check |
      |       |       |       |    |

FACTURAS.DB | PERIODO    | TOTAL |
      | Check >= _EG04 | Check |
      | <= _EG05   |    |

EDOFACT.DB | CVE_ESTADOF | EDO_FACTURA |
      |_EG03    | Check    |
EndQuery
```

El reporte que se genera por pantalla una vez que se realiza el query se presenta en la figura 4.47 y el reporte por impresora al final del capítulo.

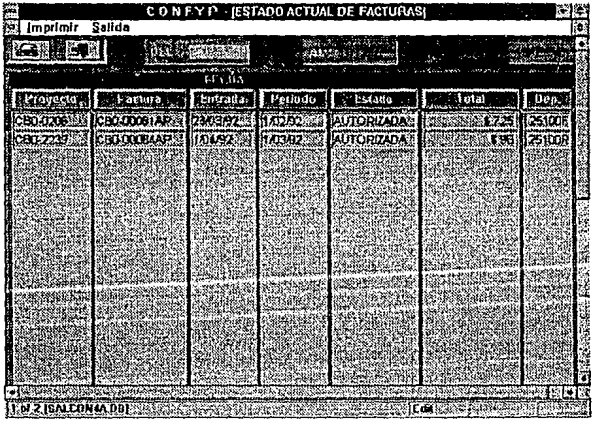


Figura 4.47 Reporte control de facturas

Las ventajas del reporte control de facturas son las siguientes: se observan todos los proyectos con sus respectivas facturas, así como la fecha, período, estado y total de cada factura, de acuerdo al rango de fechas (fecha de inicio y fecha de término) que se desee.

Es más fácil por medio de este reporte de control de facturas, llevar el control de todas las facturas que llegan en un determinado período, saber cual es el monto de cada una de ellas y que proyectos se asignan a cada factura, y posiblemente de alguna manera indirecta si lo facturado de un proyecto se excede.

Información de la factura:

Factura
Clave proyecto
Clave factura
Fecha de entrada
Período
Estado
Total
Dependencia

Estado Actual de Facturas

Para el reporte estado actual de facturas el diagrama a bloques se presenta en la figura 4.48 y el de flujo en la figura 4.49.

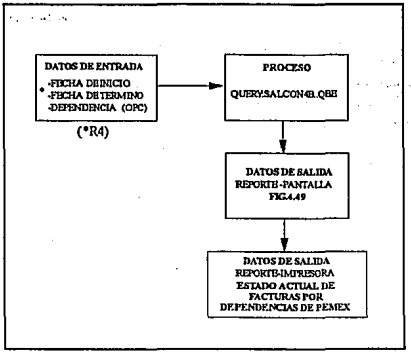


Figura 4.48 Diagrama a bloques del reporte estado actual de facturas por Dependencias de Pemex

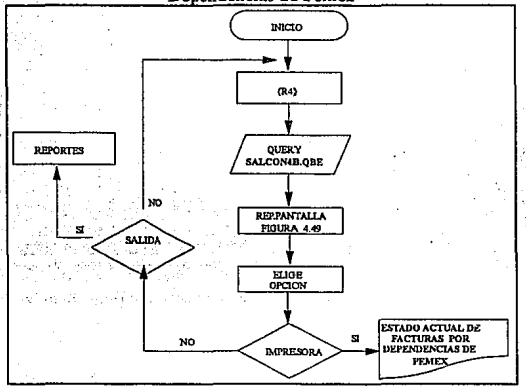


Figura 4.49 Diagrama de flujo del reporte estado actual de facturas por Dependencias de Pemex

```
QUERY PARA GENERAR LA TABLA DE SALIDA PARA EL REPORTE 4B
Query

ANSWER: :WORK:SALCON4B.DB

SORT: FACTURAS.DB->"CVE_PROYECTO", FACTURAS.DB->"CVE_FACTURA",
FECHFACT.DB->"FECHA_ENTRADA", SALIDARE.DB->"FECHA_INICIAL",
SALIDARE.DB->"FECHA_FINAL", FACTURAS.DB->"FECHA", FACTURAS.DB->"PERIODO",
FACTURAS.DB->"TOTAL", EDOFACT.DB->"EDO_FACTURA", DEPENDEN.DB->"DEPENDENCIA",

SALIDARE.DB | FECHA_INICIAL | FECHA_FINAL | CVE_DEPENDENCIA |
      | Check _EG05  | Check _EG06 | _EG04      |

FACTURAS.DB | CVE_FACTURA | CVE_PROYECTO | CVE_ESTADOF | FECHA | PERIODO |
      | Check _EG03 | Check _EG02 | _EG01    | Check | Check  |

FACTURAS.DB | TOTAL |
      | Check |

EDOFACT.DB | CVE_ESTADOF | EDO_FACTURA |
      | _EG01    | Check    |

FECHFACT.DB | CVE_PROYECTO | CVE_FACTURA | FECHA_ENTRADA | FECHA_SALIDA |
      | _EG02    | _EG03    | Check >= _EG05 | <= _EG06    |

FECHFACT.DB | CVE_DEPENDENCIA | BANDERA |
      | _EG04      | = "X"  |

DEPENDEN.DB | CVE_DEPENDENCIA | DEPENDENCIA |
      | _EG04      | Check    |
EndQuery
```

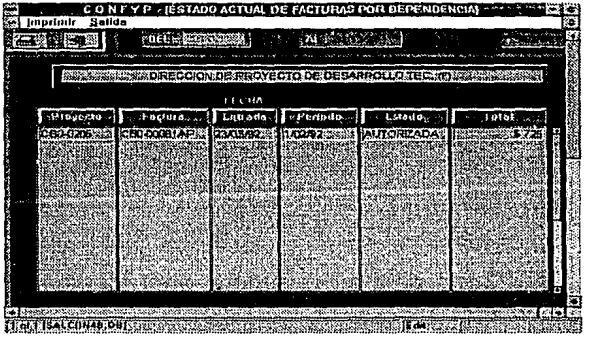


Figura 4.50 Pantalla reporte estado actual de facturas por Dependencias de Pemex

El reporte de estado actual de facturas, se puede obtener para un período determinado y también para una Dependencia específica si así se desea. El reporte que se presenta en la figura 4.50 obtiene las facturas que pertenecen a cada una de las Dependencias de Pemex, indicando el

proyecto al que pertenecen, así como la fecha de entrada, período que ampara, estado y monto total por cada factura. De esta forma cada Dependencia controla todas las facturas que están a su cargo, así como los pagos que se realizan a cada una de las facturas.

Estado Financiero de Proyectos

Para el caso del reporte denominado **Estado Financiero de Proyectos**, como se índico al inicio del capítulo, se obtiene por ocho agrupamientos diferentes y un período o sólo por un período, y la información que presenta (proyecto, tipo, renglón, estado, presupuesto, erogado etc.), es la misma para todos los reportes, sólo que se presenta de acuerdo al agrupamiento que se seleccione, el cual puede ser por Gerencia, Subgerencia, Empresa etc., todos los datos de los proyectos son importantes, tanto como para las Gerencias, Subgerencias, Empresas etc. y por ello en muchas ocasiones se requiere que este reporte genere la información perteneciente a una determinada Gerencia o a una Subgerencia o a ambas. El diagrama a bloques es el mismo para cada uno de ellos, así como el diagrama de flujo, por esta razón sólo se presenta la documentación del reporte de estado financiero de proyectos total, donde los datos de entrada determinan que datos de salida se generan. El diagrama a bloques se presenta en la figura 4.51, y el diagrama será el mismo cuando se quiera por un agrupamiento determinado (Gerencia, Subgerencia etc.) sólo que se deberá ingresar como dato de entrada. El diagrama de flujo se presenta en la figura 4.52.

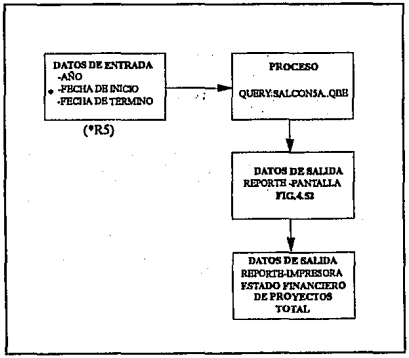


Figura 4.51 Diagrama a bloques del reporte estado financiero de proyectos total

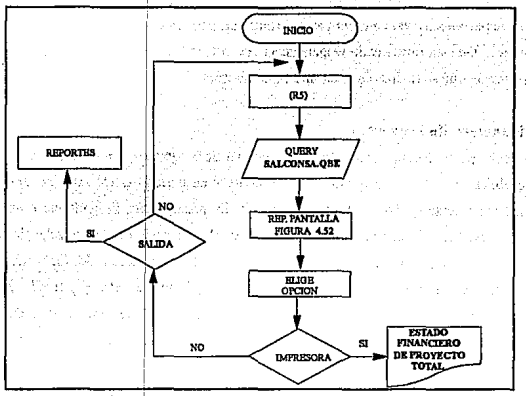


Figura 4.52 Diagrama de flujo del reporte de estado financiero de proyectos total

```
QUERY PARA GENERAR LA TABLA DE SALIDA PARA EL REPORTE 5A
Query

ANSWER: :WORK:SALCONSA.DB

SALIDARE.DB|AÑO      |FECHA_INICIAL|FECHA_FINAL|
      |Check _EG01|Check _EG03 |Check _EG04|

PROYECTO.DB|CVE_PROYECTO|CVE_TIPO|CVE_ESTADOP|CVE_GERENCIA|
      |Check _EG02 |Check  |_EG05   |_EG06    |

PROYECTO.DB|CVE_SUBGERENCIA|CVE_EMPRESA|FECHA_INICIO|FECHA_TERMINO|
      |_EG07       |_EG08    |<=_EG04   |>=_EG03    |

PROYECTO.DB|RENGLON|
      |Check |

EDOPROY.DB|CVE_ESTADOP|EDO_PROYECTO|
      |_EG05    |Check    |

PRESUPUE.DB|CVE_PROYECTO|AÑO  |CANTIDAD|
      |_EG02     |_EG01|Check  |

FACTURAS.DB|CVE_PROYECTO|CVE_ESTADOF|PERIODO       |TOTAL  |
      |_EG02     |A OR E OR R|>=_EG03,<=_EG04|CALC SUM|

GERENCIA.DB|CVE_GERENCIA|GERENCIA|
      |_EG06     |Check  |

SUBGEREN.DB|CVE_SUBERENCIA|SUBGERENCIA|
      |_EG07     |Check    |

EMPRESAS.DB|CVE_EMPRESA|EMPRESA|
      |_EG08     |Check  |
EndQuery
```

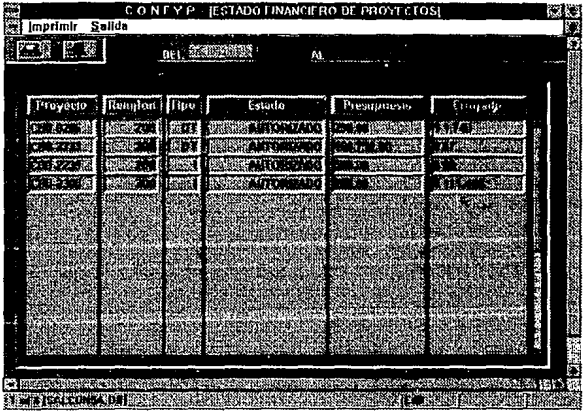


Figura 4.53 Reporte de estado financiero de proyectos total

En la Figura 4.53 se presenta la pantalla del reporte estado financiero de proyectos total, la información que se presenta es de acuerdo a los datos de entrada, los cuales como se observa en el diagrama a bloques son el año, fecha de inicio y la fecha de término, lo cual marca un rango de la información que se debe presentar, es decir la información que se presenta es la de los proyectos desarrollados en ese periodo en este caso del 31/12/92 al 1/1/94, sin importar a que Subgerencia, Gerencia o Empresa a la que pertenecen. Para obtener la información de proyectos que se desarrollan en un período determinado sólo se requiere de esos datos, pero cuando se desean esos mismos datos y además que pertenezcan a una determinada Gerencia, Subgerencia o Empresa, estos se deben de tomar como datos de entrada. El reporte es muy útil para saber en un momento dado, cuales y cuantos proyectos se desarrollan para cada Subgerencia, Gerencia etc. y además los proyectos que desarrolla cada Empresa.

El reporte como todos los demás se presenta al final del capítulo.

La figura 4.54 muestra la única gráfica que se obtiene en el sistema.

Reporte Gráfico Estado Financiero de Proyectos

El reporte gráfico que obtiene el sistema pertenece al reporte del estado financiero de proyectos, donde las barras de la figura 4.54 representan el costo, presupuesto y lo facturado de un proyecto específico, en un periodo determinado.

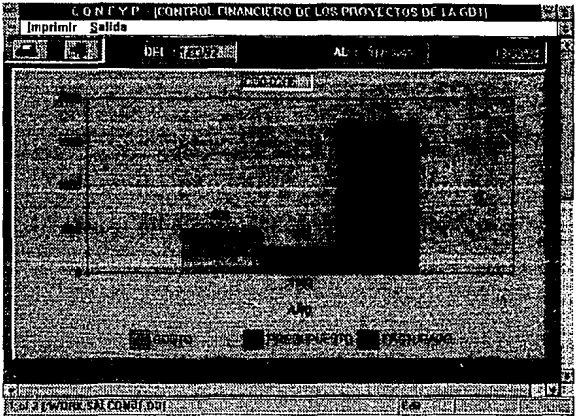


Figura 4.54 Reporte gráfico del control financiero por proyecto

Salida del sistema

Esta opción permite salir del sistema **CONFYP** y regresar a Windows

Notas:

En el proceso de captura de información en todos los módulos existen ciertas validaciones(formatos, claves únicas, campos nulos, etc.), ya sea por requerimiento del sistema o por requerimiento del usuario estas validaciones van seguidas de un mensaje informando al usuario si existe un error en la captura, indicando al usuario la forma de solucionarlo.

REPORTES DEL SISTEMA CONTROL DE FACTURAS Y PROYECTOS

CONTROL FINANCIERO DE LOS PROYECTOS

AÑO : 1992

PROYECTO : CB0-0206 **PRESUPUESTO :** $ 290

INICIA : 1/01/92 **TERMINA :** 31/12/92

TIPO : DESARROLLO TECNOLOGICO **ESTADO :** AUTORIZADO

NOMBRE : EFECTOS DE INERCIA EN PRUEBAS DE PRESION EN YACIMIENTOS FRACTURADOS

GERENCIA : GERENCIA DE DESARROLLO TECNOLOGICO

SUBGERENCIA : SUB. DE ADMINISTRACION DE YACIMIENTOS

EMPRESA : INSTITUTO MEXICANO DEL PETROLEO

RESPONSABLE : CARDENAS PLAZA OSCAR G.

PERIODO	FACTURA	ESTADO	TOTAL
1/03/92	CB0-00081AP	AUTORIZADA	$ 102
1/07/92	CB0-00705AY	AUTORIZADA	$ 1,015
1/04/92	CBF00456AY	AUTORIZADA	$ 700
1/05/92	CBF00458AY	AUTORIZADA	$ 190
1/06/92	CBF00567AY	AUTORIZADA	$ 180

CONTROL FINANCIERO DE LAS FACTURAS

FACTURA : CB0-00081AP **PROYECTO :** CB0-0206

FECHA : 19/03/92 **PERIODO :** 1/03/92

ESTADO : AUTORIZADA **TOTAL :** $ 102

CONCEPTO : PAGO PARA EL MES DE MARZO

ENTRADA	SALIDA	DEPENDENCIA
20/03/92	20/03/92	DIRECCION DE PROYECTO DE DESARROLLO TEC.
21/03/92	21/03/92	SUB. ADMINISTRACION DE YACIMIENTOS
22/03/92	22/03/92	DIRECCION DE PROYECTO DE DESARROLLO TEC. (R)
23/03/92	23/03/92	DIRECCION DE PROYECTO DE DESARROLLO TEC. (F)

PEMEX

PETROLEOS MEXICANOS

GERENCIA DE DESARROLLO TECNOLOGICO

CONTROL DE FACTURAS

06/21/94

Pág: 1

DEL : 1/01/92 AL : 31/12/92

PROYECTO	FACTURA	FECHA	PERIODO	ESTADO	TOTAL
CB0-0206	CB0-00081AP	19/03/92	1/03/92	AUTORIZADA	$ 102
CB0-0206	CB0-00705AY	26/08/92	1/07/92	AUTORIZADA	$ 1,015
CB0-0206	CBF00456AY	3/04/92	1/04/92	AUTORIZADA	$ 700
CB0-0206	CBF00458AY	5/05/92	1/05/92	AUTORIZADA	$ 190
CB0-0206	CBF00567AY	9/06/92	1/06/92	AUTORIZADA	$ 180
CB0-2232	CB000890AI	9/01/92	1/01/92	AUTORIZADA	$ 500
CB0-2233	CB0-00495AP	26/06/92	1/05/92	AUTORIZADA	$ 67
CB0-2239	CB0-00084AP	24/03/92	1/03/92	AUTORIZADA	$ 98
CB0-2300	CB0-00569AP	15/07/92	1/08/92	AUTORIZADA	$ 111,466
CBC-2102	CBC000512AY	1/01/92	1/01/92	DEVUELTA	$ 322,500
CBC-2181	CBC000343AY	13/01/92	1/03/92	DEVUELTA	$ 3,271
CBC-3100	CBC000732AY	1/01/92	1/03/92	AUTORIZADA	$ 100
CBC-3163	CBC000548AY	10/06/92	1/05/92	AUTORIZADA	$ 288,264
CBC-4164	CBC000506	20/01/92	1/03/92	AUTORIZADA	$ 58,500
CBD-2101	CBD000221AI	26/02/92	1/01/92	AUTORIZADA	$ 1,508
CBD-2101	CBD000403AI	1/02/92	1/02/92	AUTORIZADA	$ 6,213
CBD-3172	CBD000456AY	1/01/92	1/01/92	AUTORIZADA	$ 490
CBD-3198	CBD000568AY	1/02/92	1/02/92	AUTORIZADA	$ 600
CBF-3124	CBF000467AI	14/02/92	1/03/92	AUTORIZADA	$ 666
CBF-3124	CBF000559AI	13/05/92	1/01/92	AUTORIZADA	$ 1,847
CBF-3124	CBF000604AI	12/05/92	1/04/92	AUTORIZADA	$ 6,005
CBF-3124	CBF000694AI	8/06/92	1/06/92	AUTORIZADA	$ 6,772
CBF-3127	CBF000485AY	27/04/92	1/03/92	AUTORIZADA	$ 7,254
CBF-3127	CBF000612AY	12/05/92	1/04/92	AUTORIZADA	$ 68,253
CBF-3127	CBF000721AY	8/06/92	1/06/92	AUTORIZADA	$ 73,024
CBI-5240	CBI000234AY	3/01/92	1/01/92	AUTORIZADA	$ 567

NUM. FACTURAS : 26

NUM. FACTURAS : 26

ESTADO ACTUAL DE FACTURAS

DEL : 1/01/92 AL : 31/12/92

DIRECCION DE PROYECTO DE DESARROLLO TEC.

PROYECTO	FACTURA	FECHA ENT.	PERIODO	ESTADO	TOTAL
CB0-2233	CB0-00495AP	28/06/92	1/05/92	AUTORIZADA	$ 67
CB0-2300	CB0-00569AP	16/07/92	1/06/92	AUTORIZADA	$ 111,466
CB0-0206	CB0-00705AY	28/08/92	1/07/92	AUTORIZADA	$ 1,015
CB0-2232	CB000890AI	10/01/92	1/01/92	AUTORIZADA	$ 500
CBC-2181	CBC000343AY	15/03/92	1/03/92	DEVUELTA	$ 3,271
CBC-4164	CBC000506	22/04/92	1/03/92	AUTORIZADA	$ 58,500
CBC-2102	CBC000512AY	2/01/92	1/01/92	DEVUELTA	$ 322,500
CBC-3163	CBC000548AY	11/06/92	1/05/92	AUTORIZADA	$ 288,264
CBC-3100	CBC000732AY	2/01/92	1/03/92	AUTORIZADA	$ 100
CBD-2101	CBD000403AI	2/02/92	1/02/92	AUTORIZADA	$ 6,213
CBD-3172	CBD000456AY	2/01/92	1/01/92	AUTORIZADA	$ 490
CBD-3198	CBD000568AY	2/02/92	1/02/92	AUTORIZADA	$ 600
CBF-3124	CBF000467AI	15/04/92	1/03/92	AUTORIZADA	$ 666
CBF-3127	CBF000495AY	28/04/92	1/03/92	AUTORIZADA	$ 7,254

DIRECCION DE PROYECTO DE DESARROLLO TEC. (F)

PROYECTO	FACTURA	FECHA ENT.	PERIODO	ESTADO	TOTAL
CB0-0206	CB0-00081AP	23/03/92	1/03/92	AUTORIZADA	$ 102

DIRECCION DE PROYECTO DE DESARROLLO TEC. (R)

PROYECTO	FACTURA	FECHA ENT.	PERIODO	ESTADO	TOTAL
CB0-2239	CB0-00084AP	1/04/92	1/03/92	AUTORIZADA	$ 98

NUM. FACTURAS: 16 TOTAL: $ 801,106

PETROLEOS MEXICANOS 06/21/94

GERENCIA DE DESARROLLO TECNOLOGICO Pág : 1

ESTADO FINANCIERO DE PROYECTOS

DEL: 1/01/92 AL : 31/12/92

GERENCIA DE DESARROLLO TECNOLOGICO

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CB0-0206	DT	208	AUTORIZADO	290,00	2.186,50
CB0-2232	DT	208	AUTORIZADO	256.754,00	500,00
CB0-2233	DT	208	AUTORIZADO	166.754,00	67,00
CB0-2239	I	208	AUTORIZADO	500,00	98,00
CBC-3100	DT	308	AUTORIZADO	1.750,00	100,00
CBC-4164	DT	308	AUTORIZADO	100,00	58.500,00
CBD-2101	DT	308	AUTORIZADO	615,00	7.721,00
CBF-3124	S	208	AUTORIZADO	57,00	15.290,00
CBF-3127	DT	208	AUTORIZADO	450,00	148.531,00
CBI-5240	S	208	AUTORIZADO	120,00	567,00

EMPRESA : INSTITUTO MEXICANO DEL PETROLEO

GERENCIA : GERENCIA DE DESARROLLO TECNOLOGICO

SUBGERENCIA : SUB. DE ADMINISTRACION DE YACIMIENTOS

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CB0-0208	DT	208	AUTORIZADO	290,00	2.186,50
CB0-2233	DT	208	AUTORIZADO	166.754,00	67,00
CB0-2239	I	208	AUTORIZADO	500,00	98,00

SUBGERENCIA : SUB. DE APLICACION Y EVAL.DE NUEVAS TECNICAS

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CB0-2232	DT	208	AUTORIZADO	256.754,00	500,00

SUBGERENCIA : SUB. DE DESARROLLO PROFESIONAL

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBF-3124	S	208	AUTORIZADO	57,00	15.290,00

ESTADO FINANCIERO DE PROYECTOS

DEL : 1/01/92 AL : 31/12/92

EMPRESA : INSTITUTO MEXICANO DEL PETROLEO

GERENCIA : GERENCIA DE DESARROLLO TECNOLOGICO

SUBGERENCIA : SUB. SIMULACION DE PROCESOS DE PRODUCCION

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBC-3100	DT	308	AUTORIZADO	1.750,00	100,00
CBC-4164	DT	308	AUTORIZADO	100,00	58.500,00

SUBGERENCIA : SUB. SISTEMAS DE PROD. Y MANEJO DE H.C.

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBD-2101	DT	308	AUTORIZADO	615,00	7.721,00

SUBGERENCIA : UNIDAD DE COMPUTACION APLICADA

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBF-3127	DT	208	AUTORIZADO	450,00	148.531,00
CBI-5240	S	208	AUTORIZADO	120,00	567,00

EMPRESA : INSTITUTO MEXICANO DEL PETROLEO

GERENCIA : GERENCIA DE DESARROLLO TECNOLOGICO

GERENCIA : GERENCIA DE PROGRAMACION Y EVALUACION

SUBGERENCIA : COMPUTACION APLICADA

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CB0-2300	I	308	AUTORIZADO	200,00	111.466,00

SUBGERENCIA : GERENCIA DE DESARROLLO TECNOLOGICO

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBC-3163	S	208	AUTORIZADO	476,00	288.264,00

SUBGERENCIA : GERENCIA DE PROGRAMACION Y EVALUACION

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBD-3172	DT	308	AUTORIZADO	488,00	490,00

EMPRESA : INSTITUTO MEXICANO DEL PETROLEO

GERENCIA : GERENCIA DE PROGRAMACION Y EVALUACION

SUBGERENCIA : GERENCIA DE PROGRAMACION Y EVALUACION

SUBGERENCIA : SUB. SIMULACION DE PROCESOS DE PRODUCCION

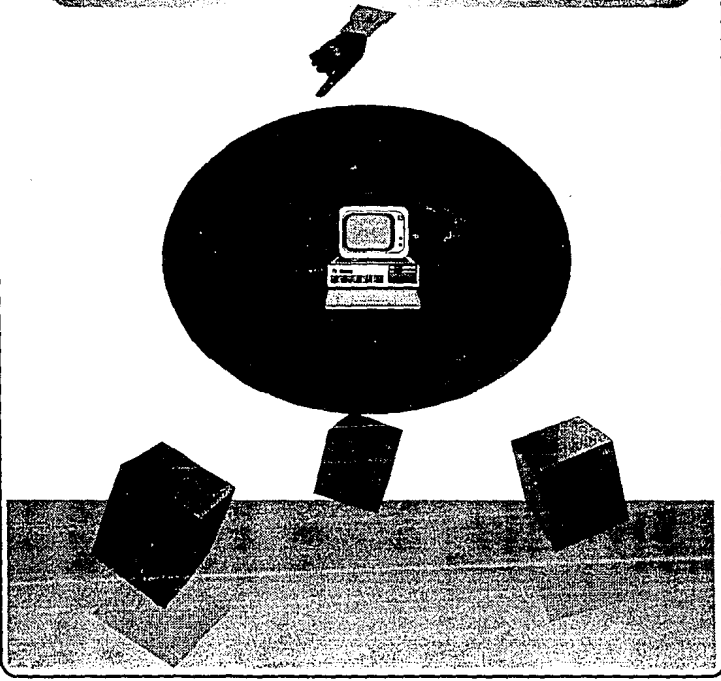
PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CBD-3198	DT	308	AUTORIZADO	600,00	600,00

SUB. DE ADMINISTRACION DE YACIMIENTOS

PROYECTO	TIPO	RENGLON	ESTADO	PRESUPUESTO	EROGADO
CB0-0206	DT	208	AUTORIZADO	290,00	2.186,50
CB0-2233	DT	208	AUTORIZADO	166.754,00	67,00
CB0-2239	I	208	AUTORIZADO	500,00	98,00

CAPITULO V

Panorámica de la Orientación a Objetos



PANORAMICA DE LA ORIENTACION A OBJETOS

El avance de las técnicas de diseño de sistemas computacionales (software), comparado con el de hardware, es menos impresionante. Las causas comúnmente mencionadas son: la poca reutilización de piezas de software en nuevos sistemas, pues generalmente, cuando se realiza el diseño de algún sistema, del mismo modo se realiza la construcción de nuevo software implicando la inversión de una gran cantidad de dinero, esfuerzo, tiempo y sobretodo productividad. Esto provoca que cada vez que se desarrolle un nuevo sistema se invierta tiempo, incumplimiento en las fechas de terminación y como consecuencia sobrecostos.

Otra de las causas es la ***composición de programas complejos y difíciles de manejar***, así como ***el propio método de diseño estructurado de sistemas.*** El diseño estructurado está basado en la descomposición jerárquica del sistema, alrededor de las operaciones (algoritmos), mientras que los datos se encuentran en un espacio común compartido por las operaciones. Este tipo de organización funciona bien mientras los sistemas tengan un tamaño moderado y sean desarrollados individualmente. Los sistemas actuales son mucho más complejos y es indispensable su construcción por un grupo de personas, utilizar el diseño estructurado en este tipo de sistemas trae como desventajas que las decisiones iniciales de diseño en los niveles superiores puedan tener que reconsiderarse cuando el diseño progresa hacia niveles inferiores esto puede requerir un encadenamiento hacia atrás en el diseño y una reescritura considerable de código, además pueden resultar muy caras las pruebas en el sistema para procedimientos nuevos que se van añadiendo de la manera deseada. Es por ello que en este capítulo, se presentan algunas de las soluciones para los problemas antes mencionados, así como ventajas, desventajas, desarrollo de aplicaciones y tendencias futuras de la orientación a objetos.

5.1 PROGRAMACION ORIENTADA A OBJETOS EN LA INGENIERIA DE SOFTWARE

La programación orientada a objetos induce un cambio en la cultura computacional, que va mucho más allá de la incorporación de nuevos mecanismos de abstracción en los lenguajes de programación. En esta nueva perspectiva, el desarrollo de sistemas es una actividad que consiste en generar componentes que puedan ser reutilizables. Un componente es un módulo cuya interfaz con el exterior está claramente especificada. La descripción enfatiza **(Qué)** es lo que hace el componente, presentando la información relevante y ocultando los detalles que son irrelevantes para el entendimiento de la abstracción. Un sistema se constituye en base a componentes previamente diseñados e implantados. Los componentes se seleccionan de una biblioteca y se especializan de acuerdo a requerimientos concretos. El objetivo final, a largo plazo, es amortizar el costo de desarrollo de sistemas mediante la reutilización de componentes. Esta perspectiva cambia, también, el ciclo de desarrollo de un sistema, así como las metodologías de análisis y diseño, la administración de sistemas y, en general, la cultura de desarrollo de sistemas. El desarrollo de sistemas con la metodología orientada a objetos tiene un doble propósito: por un lado la elaboración de herramientas que solucionen la problemática planteada y, por otro, la generación de componentes, aumentando así el software. El impacto económico de la adopción de la metodología orientada a objetos no puede medirse en base a la construcción de un solo sistema; la ganancia del uso de esta metodología se muestra a largo plazo. Un estudio elaborado por los laboratorios AT&T de Estados Unidos, reportó que un 80% de código de aplicaciones nuevas, en el área de diseño de redes de computadoras, fue reutilizado después de cuatro años de ser adoptada la metodología orientada a objetos.

5.1.1 Reusabilidad

La reusabilidad del software no es un objetivo establecido recientemente , ni es privado de la programación orientada a objetos. Desde 1968 en la conferencia de OTAN, sobre la crisis de producción de software, se planteó la necesidad de generar bibliotecas de componentes para facilitar la producción del software. La programación orientada a objetos ofrece mecanismos que apoyan la reusabilidad. En particular tiene mecanismos para definición (clases y objetos), especialización (herencia) etc. De igual importancia para lograr la reusabilidad son las metodologías de análisis y diseño (estudiadas en el capítulo II), las cuales promueven la generación de componentes reutilizables y generan modelos con un alto nivel de estructuración.

El principal problema al que se enfrenta la reusabilidad de componentes, es la especificación semántica de los mismos. Para poder usar un componente es indispensable entender antes qué es lo

que hace. La interfaz de un componente contiene sólo información sintáctica acerca de los nombres y tipos de los servicios disponibles, pero no incluyen información acerca del significado operacional de cada uno de ellos. La base para la reusabilidad es una buena biblioteca de componentes reutilizables. La biblioteca de componentes debe facilitar el proceso de reutilización de componentes proyectando:

- Un buen nivel de abstracción que permita entender mejor qué hace un componente.
- Mecanismos de selección para localizar, comparar y seleccionar componentes.
- Mecanismos de especialización de componentes usando parámetros, transformaciones, restricciones o algún otro tipo de refinamiento.
- Mecanismos de integración para combinar la colección de componentes seleccionados en un sistema.

5.2 ALCANCES

Desde sus inicios el software se ha visto como una forma de hacer que la computadora lleve a cabo una tarea particular. En todas las ocasiones el resultado es un sistema que realiza la tarea original, pero que es inapropiado para manejar otras tareas. Por ejemplo un sistema de facturación, no realiza otra actividad que no sea sino la de controlar las comisiones para un grupo de ventas.

Para construir sistemas, el enfoque orientado a objetos tiene muchas ventajas además de flexibilidad. Si se tiene un modelo adecuado de los clientes de una organización y sus interacciones con ellos, dicho modelo puede usarse lo mismo para facturar que para lista de correo, que para pagar comisiones. Como la estructura del software refleja el mundo real con mayor precisión, las operaciones básicas de cualquier organización tienden a cambiar más lentamente que las necesidades de grupos, lo que significa que el software basado en modelos corporativos vivirá tanto como la organización misma, evolucionando con ella.

De lo anterior se observa que el proceso por el cual el software se construye es muy diferente en el enfoque orientado a objetos. Los objetos al ser bloques generales que modelan entidades reales, más que realizar tareas específicas permiten reutilizarlos en proyectos subsecuentes aún si los objetivos del nuevo proyecto son totalmente diferentes. Entre más clases se acumulen, el esfuerzo de desarrollar software será cada vez más aplicado en ensamblar objetos existentes en nuevas formas.

5.3 BENEFICIOS

La alternativa que ofrece el desarrollo de sistemas con tecnología de orientación a objetos es atractiva porque, en primer nivel, propicia expresar la solución a los problemas de proceso de información en el mismo lenguaje y con los mismos objetos que componen dicho problema, en un segundo nivel, la solución generalmente se implanta utilizando, siempre que sea posible, objetos existentes, ahorrando tiempo y esfuerzo al no escribir código, y al mismo tiempo, aumentar la calidad por utilizar objetos ya probados.

La unión de información y acciones en objetos, y la comunicación entre ellos, expresada como mensajes, es apropiada para desarrollar en forma escalable sistemas complejos y de gran tamaño, creando nuevas estructuras activas de información que puedan iniciar acciones a partir de la información que contenga.

5.4 DESVENTAJAS

Actualmente existen muchas y diversas metodologías de análisis y diseño, aunque se captan algunos signos de convergencia principalmente hacia lo que se basa en el componente de los modelos empresariales.

- Los lenguajes de programación orientados a objetos están en proceso de estandarizarse, notablemente C++ y SMALLTALK. Sin embargo, persisten problemas de portabilidad en la mayoría de los lenguajes e incompatibilidad entre diferentes proveedores. Esto impacta también a los proveedores de bases de datos orientadas a objetos, que soportan dos y en raros casos tres diferentes proveedores de lenguajes.

- Las herramientas han mejorado notablemente, sin embargo se requiere perfeccionarlas para que asistan en el diseño de objetos y sus interrelaciones, que faciliten la administración y el acceso a bibliotecas gráficas para capturar información y generar reportes.

- El costo de la conversión asociada con la transición a esta tecnología es significativo. El costo más obvio es la adquisición del nuevo software: lenguajes, bases de datos y herramientas que permitan aplicar la tecnología.

- Otro aspecto es en el hardware, debido a que la mayoría de los sistemas orientados a objetos utilizan gráficas de manera intensiva y las Empresas que utilizan principalmente terminales de caracteres para acceder minicomputadoras requieren invertir en máquinas tipo estación de trabajo o computadoras personales, creando arquitecturas **cliente/servidor**.

- Por otro lado, la educación y capacitación de administradores y técnicos es también costosa, y los programadores seguramente bajen su productividad durante el aprendizaje de las nuevas técnicas.

- Además un costo que frecuentemente no se toma en consideración es el de crear bibliotecas de clases reutilizables y el de crear modelos, también reutilizables. Este es de hecho, el mayor costo de conversión, sin embargo, es también el que mejor retorno de inversión ofrece, porque crea una plataforma para desarrollar nuevas aplicaciones más rápido.

5.5 MIGRACION HACIA LA TECNOLOGIA ORIENTADA A OBJETOS

La tecnología orientada a objetos comienza a abrirse paso entre la comunidad empresarial. Sus beneficios potenciales, tales como el incremento en la productividad de los desarrolladores de sistemas y la producción de sistemas más flexibles, pueden constituirse en un factor decisivo de la estrategia tecnológica de la Empresa. Dado que la adopción de este paradigma representa un cambio radical en el método de trabajo de la organización, este cambio trae consigo una fuerte inversión. Además de la inversión inicial en nuevas herramientas, se incurre en costos (ver figura 5.1) de capacitación, reorganización y adaptación cultural.

NOMBRE	COSTO	DURACION
Conceptos Fundamentales de la Metodología de Objetos.	N$ 1,600.00 + IVA	12 Horas
Taller de Introducción a la Programación Orientada a Objetos.	N$ 1,200.00+IVA	20 Horas
Taller de Programación Orientada a Objetos en C++.	N$ 1,200.00+IVA	20 Horas
Taller de Programación Orientada a Objetos en CLIPPER 5.0.	N$ 1,200.00 + IVA	20 Horas
Taller de Actor, Ambiente de Programación Orientado a Objetos.	N$ 1,200.00 + IVA	20 Horas
Diseño de Sistemas con la Metodología de Objetos.	N$ 1,600.00 + IVA	12 Horas

Figura 5.1 Costos de cursos intensivos impartidos en la "Fundación Arturo Rosenbluetr" durante los meses de Febrero a Junio de 1993

5.5.1 Consideraciones de implantación

La implantación de un nuevo paradigma de desarrollo presenta un cambio radical en el método de trabajo del departamento de sistemas y de la Empresa en general, por ello es necesario estar consiente del costo involucrado en el cambio.

Fases del proceso

El proceso de asimilación de toda innovación en una Empresa pasa por varias etapas antes de que se pueda considerar que el cambio esté totalmente integrado:

a) La fase de reestructuración, durante la cual la herramienta y la organización que utiliza se transforman para adaptarse mutuamente.

b) La fase de estabilización, durante la cual la herramienta es aceptada y su utilidad reconocida por todos los involucrados en su utilización.

c) La fase de integración, es la etapa final en la utilización de la herramienta ya que forma parte de la rutina habitual de la Empresa.

La separación entre estas fases puede ser no muy clara y presentar algunos traslapes. La fase más notoria es la primera, ya que se acompaña de la introducción de la nueva tecnología y del cambio de hábitos de trabajo.

Fase de Reestructuración

Durante esta fase se realiza la adaptación de la tecnología para que satisfaga las necesidades específicas de la Empresa, y se modifican la estructura y los procedimientos organizacionales para que se propicie un mejor ambiente de aplicación de la nueva tecnología. Los factores que se consideran críticos para una implantación exitosa se puede clasificar en tres grupos:

1) Atributos propios de la tecnología

a) **Dificultad de aprendizaje.** La tecnología orientada a objetos necesita una cantidad considerable de entrenamiento. El desarrollo se confronta con una serie de conceptos nuevos, tales como herencia, el polimorfismo, etc.

b) **Observabilidad de resultados.** Los beneficios de la orientación a objetos (reutilización, abstracción, facilidad de mantenimiento, etc.) son prácticamente intangibles. Sólo un riguroso registro de métricas de software puede demostrar su validez. Entre las métricas que se pueden recopilar en sistemas de objetos están:

Tiempo de diseño
Tiempo de implantación
Costo unitario
Reutilización de componentes: cantidad, costo,
Costo de entrenamiento
Código reutilizable generado.

2) Atributos propios de la organización

a) **Actitud del personal de sistemas.** Quien pueda actuar defensivamente ante la exigencia de aprender un nuevo método de trabajo. Para evitar que este factor tenga una influencia negativa sobre la transferencia de la tecnología se les deberá manifestar claramente el soporte de la Gerencia, además se les debe ofrecer un amplio programa de capacitación y se debe considerar el tiempo necesario para adaptarse.

3) Atributos de la interacción tecnología-organización

a) **Apoyo de la alta dirección.** Se debe sentir el apoyo total de los altos directivos independientemente de quien haya tomado la decisión tecnológica. Esta es una condición necesaria para la asimilación de un nuevo método de trabajo, y este apoyo deberá manifestarse en forma de una gran disponibilidad de recursos de toda índole: motivaciones, financieras e institucionales.

b) **Estrategia de implantación.** Implica la existencia de un plan sistemático y bien difundido.

5.5.2 Estrategias

Existen tres posibles estrategias de la metodología:

Vertical. Esta es la más utilizada en la práctica y consiste en realizar proyectos donde se llegue a utilizar completamente la herramienta. Generalmente se empieza por proyectos piloto y corto alcance y después se aplica a proyectos generalmente de la Empresa. La ventaja de este enfoque es que la experiencia de un proyecto se aprovecha en los siguientes (lo que justifica la necesidad de avanzar progresivamente de proyectos simples a otros más complejos).

Horizontal. En esta estrategia se propone que la tecnología sea introducida por fases, por ejemplo se enseña la fase de análisis a todos y después la fase de diseño. Esta estrategia tiene la desventaja de ser muy lenta.

Combinación. En ésta todos empiezan al mismo tiempo, algunos por fase y otros completamente en un proyecto piloto.

En el caso de los sistemas orientados a objetos, la naturaleza del paradigma sugiere una estrategia vertical ya que el proyecto se fundamenta en los proyectos que le preceden. A pesar de ello, algunas de las ventajas que presenten los sistemas orientados a objetos no pueden alcanzarse desde el desarrollo, por lo siguiente:

a)Los desarrolladores no tienen experiencia suficiente en la nueva forma de trabajo.

b)No hay ningún componente que reutilizar.

c)Diseñar componentes reutilizables requiere de un esfuerzo adicional que alarga esta etapa.
Esto lleva a proponer una estrategia basada en el desarrollo de varios proyectos de manera que cada uno de ellos permita demostrar alguna de las características del nuevo paradigma. Los primeros proyectos sirven para demostrar el potencial de los objetos para construir modelos de la Empresa, permitiendo además que el personal se capacite y desarrolle objetos reutilizables. El último proyecto servirá para comparar las métricas obtenidas durante el desarrollo con los estándares que se obtenían con los métodos anteriores. Este proceso es relativamente largo pero evita las frustraciones que se presenten cuando se esperan resultados inmediatos. Algunas Empresas han abandonado el paradigma después de haber pasado varios meses desarrollando un proyecto complicado, perdiendo así lo que habían ganado en experiencia y en componentes reutilizables, y comprometiendo para siempre la credibilidad de los objetos en dicha compañía.

5.5.3 Preservación y evolución de los sistemas existenciales

Toda Empresa en la actualidad cuenta con un acervo de sistemas que no pueden ser descartados de forma rápida y sobretodo desarrollarse nuevamente usando técnicas orientadas a objetos. Ambas tecnologías deberán coexistir, al menos durante cierto tiempo, por ello la estrategia deberá actuar sobre 2 ejes principales:

1) Reconstruir poco a poco aquellos sistemas (o subsistemas) que vayan necesitando mantenimiento y cuya naturaleza se preste a la utilización del paradigma.

2) Encapsular los sistemas convencionales en una envolvente que simule un sistema orientado a objetos. La envolvente recibe los mensajes que provienen de otros objetos y los transforma en llamadas procedurales; al terminar la operación, la envolvente responde con los resultados correspondientes. Esta estrategia puede aplicarse a sistemas completos o bien a los módulos que los componen.

Durante el tiempo de aplicación de cualquiera de estas estrategias, los sistemas de la Empresa irán recorriendo poco a poco el espectro desde un paradigma clásico hacia uno orientado a objetos, como se puede ver en la figura 5.2.

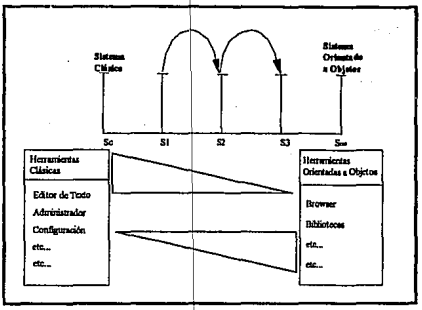
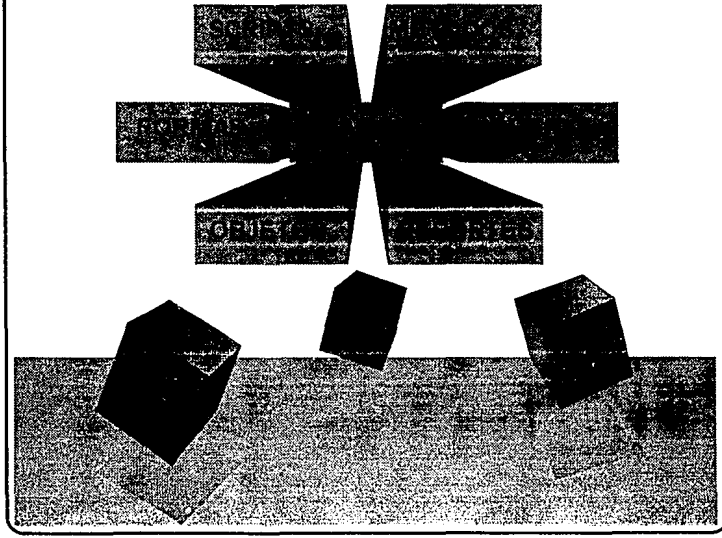


Figura 5.2 Reconstrucción incremental de sistemas

APENDICE A

FUNDAMENTOS DE PARADOX FOR WINDOWS



FUNDAMENTOS DE PARADOX FOR WINDOWS

PARADOX para Windows permite crear y trabajar con base de datos en varios formatos. Además de su formato básico PARADOX para Windows puede manejar el de PARADOX para Dos, DBASE III Plus, y DBASE IV.

Una base de datos es un conjunto de datos. Usualmente estos datos son texto o números, pero en el caso de PARADOX para Windows también se puede incluir otro tipo de información, como sonidos e imágenes. Al igual que la mayoría de las bases de datos, las de PARADOX tienen una estructura específica para tablas, registros y campos.

Términos básicos

Todos los datos en una base PARADOX están contenidos en **tablas**; cada una contiene información sobre elementos semejantes. Las bases de datos muy sencillas están formadas por una tabla, pero la mayor parte consiste en dos o más tablas **relacionadas**. Una tabla esta relacionada por **registros**; cada uno contiene información sobre un solo elemento.

Cada registro esta formado por **campos** con nombre. Cada campo contiene una pieza de información específica. El nombre de un campo describe este **valor**. Se puede ver una tabla como una matriz de información. Cuando se crea una tabla, se debe definir cada campo, dándole un nombre y un tipo (alfanumérico, numérico, etc.). La mayor parte de los campos en las tablas son alfanuméricos tienen letras y números. También se pueden definir campos que tengan solamente valores numéricos, fechas u otro tipo de información.

En PARADOX cada usuario decide el tamaño de los campos alfanuméricos, de acuerdo con los valores que se espera que contengan; este valor puede ser de 1 a 255 caracteres. Si es necesario, se cambia de manera fácil el tamaño de los campos mientras se desarrolla la base de datos.
Los tamaños de algunos tipos de campos, como los numéricos y de fecha, están preestablecidos por PARADOX. Otros tipos, por ejemplo los Memo, pueden contener casi cualquier tipo de información, limitada solamente por el espacio disponible en el disco duro.
Existen tres etapas relacionadas en la creación de una base de datos: **planeación, implementación y pruebas**. La **planeación** es lo primero. Entre más cuidadosamente se planee como se va a utilizar una base de datos, se encuentra la menor cantidad de problemas al implementarla y probarla. Sin embargo, siempre se necesita modificar el plan original conforme avanza en las etapas de implementación y pruebas. PARADOX es lo suficientemente flexible para permitir hacer esto más fácil.

Herramientas de PARADOX

Una base de datos PARADOX consiste en **objetos importantes** (tablas, formas, botones, reportes, querys, scripts, métodos, etc.) y en **objetos de diseño** (campos, textos, gráficas etc.). Cada objeto tiene un conjunto de atributos modificables llamados propiedades. Una tabla por ejemplo tiene propiedades como color o patrón de fondo, el texto tiene cierto tipo, color y tamaño, las líneas tienen cierto ancho y color. Hay algunas propiedades que todos los objetos tienen y algunas que son específicas para cierto tipo de objeto. Todas la propiedades tienen un valor preestablecido, que PARADOX utiliza a menos que se cambie el valor. PARADOX utiliza los objetos importantes para almacenar, mostrar, imprimir y manipular datos.

TIPO DE OBJETO	DESCRIPCION
Forma	La forma define relaciones entre objetos. Se utiliza para controlar el desplegado de datos en la pantalla y para esbalonar tablas.
Consulta	Las consultas son preguntas que se plantean acerca de los datos de las tablas. También se pueden utilizar consultas para insertar, borrar y editar datos en una tabla; así como para ejecutar cálculos con los datos de una tabla.
Informe	Los informes se utilizan principalmente para diseñar la manera en que los datos almacenados en tablas deben aparecer al imprimirse. Se pueden utilizar informes para ordenar y agrupar datos y también para calcular valores de datos en las tablas.

TIPO DE OBJETO	DESCRIPCION
Tabla	Las tablas contienen datos acomodados en filas y columnas. Cada fila de una tabla, denominada registro contienen información acerca de cada elemento. Cada columna denominada campo, contienen información acerca de los elementos.

Figura A.1 Objetos importantes

Objetos de Diseño

Los objetos de diseño se pueden crear en documentos utilizando las herramientas de la barra de velocidad.

TIPO DE OBJETO	DESCRIPCION
Recuadro	Las cajas son rectángulos que se pueden colocar en un documento.
Botón	Los botones son objetos que se pueden colocar en una forma (más no es un informe) para iniciar acciones. Cuando se asocia un método ObjectPAL con un botón, dicho método se ejecuta cuando se da click al botón.
Tabulaciones	Las tabulaciones presentan información en un documento derivada de tablas en formato similar al de una hoja de cálculo
Elipse	Las elipses se utilizan para llamar la atención hacia los objetos en un documento; el efecto es semejante a haber trazado un círculo alrededor de un objeto.
Campo	Los campos contienen valores derivados de una o más tablas, o valores especiales que muestran la fecha u horas actuales o el número de página en un documento. Cada campo contiene un rótulo y un valor.
Gráfica	Las gráficas son objetos que presentan datos de tablas en forma gráfica en un documento.

TIPO DE OBJETO	DESCRIPCION
Imagen	Una imagen es una representación de un campo de tipo gráfico en una tabla o una imagen gráfica independientemente en un documento. PARADOX acepta gráficas del Clipboard y también las importa de archivos de tipo .BMP, GIF, .EPS, .PCX o .TIF.
Linea	Las líneas se pueden crear en un documento para mejorar su aspecto.
Multiregistro	Un objeto multiregistro presenta un patrón repetitivo de campos en un documento. Después de definir la distribución de los campos de un objeto, un objeto multiregistro utiliza el mismo formato para mostrar los valores de varios registros.
OLE	Se puede utilizar la función de Windows llamada Object Linking and Embedding (OLE) para colocar objetos de otras aplicaciones en contenedores PARADOX.
Marco de Tabla	Un marco de tabla es un objeto que puede contener la información en una tabla. Después de colocar un marco a una tabla en un documento se puede unir a él, de manera que el marco contenga los datos de la tabla unida a él.
Texto	Los objetos de texto contienen texto de cualquier formato disponible en un documento. Los objetos de texto están contenidos dentro del marco de texto.

Figura A.2 Objetos de diseño

Cada objeto de diseño tiene algunas propiedades a las cuales PARADOX asigna valores preestablecidos, pero que se pueden modificar. La caja de propiedades, por ejemplo, contiene los colores de relleno y de patrón, y el estilo, color y grueso del marco.

Object Tree (Arboles de objetos)

Un árbol de objetos muestra relaciones entre los objetos de un documento. Conforme se vaya haciendo más compleja la base de datos, especialmente cuando se asocie código ObjectPAL a objetos, se encontrará que los árboles de objetos proporcionan información muy valiosa acerca de la estructura de los documentos, los nombres de los objetos y las relaciones de contenedor entre los objetos. Un árbol de objetos también proporciona un medio adecuado para seleccionar objetos, especialmente en documentos complejos.

Categorías de objetos

En la figura A.3 se pueden observar como los tipos de objetos son agrupados dentro de categorías y son relacionados con la aplicación de ObjectPAL.

CATEGORIA	TIPOS
Events	ActionEvent,ErriorEvent,Event,KeyEvent,MenuEvent ,MouseEvent,MouseEvent,MoveEvent,StatusEvent,Ti merEvent,ValueEvent.
Desig Objects	Menu,PopUpMenu,UIObject.
Display managers	Applicactions,Form,Report,TableView.
Data Types	AnyType,Array,Binary,Currency,Date,DateTime,Dy nArray,Graphic,Logical,LongInt,Memo,Number,OL E,Point,Record,SmallInt, String,Time.
Data model objects	Database,Query,Table, TCursor.
System data objects	DDE,FileSystem,Library,Session,System, TextStream.

Figura A.3 Categorías de objetos

Métodos

Los métodos son instrucciones que se ejecutan cuando ocurre un evento específico en el ambiente PARADOX. Cuando se inspecciona un objeto en un documento, aparece un menú con una lista de propiedades. La última de estas propiedades es Methods (Métodos). Estos métodos son esqueletos que se pueden utilizar para crear métodos particulares. Los métodos especifican como los objetos responden a los eventos.

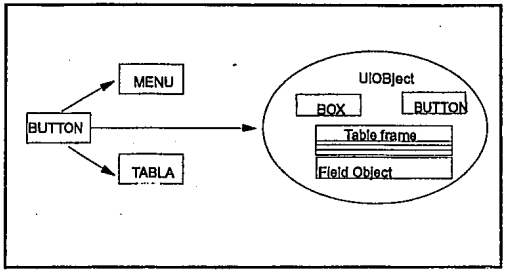


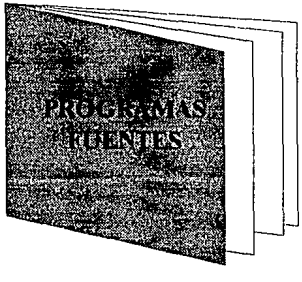
Figura A.4 Métodos y objetos

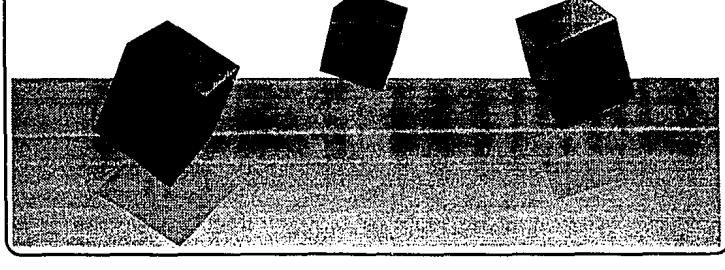
En la figura A.4 se considera el button menú y la tabla como objetos y la conjunción de ellos mediante eventos.

APENDICE B

PROGRAMAS FUENTES

CONTROL DE FACTURAS Y PROYECTOS





PROGRAMAS FUENTES DEL CONTROL DE FACTURAS Y PROYECTOS

El desarrollo del software del Sistema **Control de Facturas y Proyectos** en el manejador de base de datos **PARADOX FOR WINDOWS** se manejo de tal manera que un **script, procedimiento o método** generado para el sistema se utilice de manera constante para los módulos del sistema que sean semejantes, por ello es que en este punto sólo se muestran programas generales y concretos al lector, de forma que se ejemplifica el desarrollo de software para el sistema en **PARADOX FOR WINDOWS Versión 1.0.**

La base principal en la cual se desarrollo el sistema de **Control de Facturas y Proyectos** es aprovechar a lo máximo las características del manejador de base de datos utilizado, además de reducir tiempo y costo, esto no indica que la programación sea casi nula, ya que a pesar de que PARADOX for Windows versión 1.0 es bastante práctico y disminuye en gran medida el tiempo de desarrollo de software, existen algunos procesos que son necesarios realizar programando utilizando el *Lenguaje de Programación Integrado* **(PAL)** de PARADOX for Windows ver 1.0.

Además de hacer uso del **PAL** de PARADOX for Windows se requirió de algunas opciones de PARADOX como son:

Creación de tablas
Formas
Reportes
Querys
Script
Métodos
Procedimientos

A continuación se describen todos los scripts, métodos y procedimientos generados para el desarrollo del **CONFYP**.

1.- Entrada al sistema

Para la entrada al sistema se utiliza un script general que manda a ejecutar la forma principal (menú) para sistema de control de facturas y proyectos. A partir de este script se realizan llamados a métodos, procedimientos, querys y formas para toda la ejecución del sistema.

(1) Script que ejecuta el programa principal sistema del CONFYP

```
Proc Presenta()
Var
  PRESENTA FORM
EndVar
Message("Espere un momento. Cargando el sistema C O N F Y P.")
PRESENTA.Open("MENU1.FSL")
EndProc
Method Run(Var EventInfo Event)
  Presenta()
EndMethod
```

2.- Menú principal

Los programas fuentes para ejecutar la forma del menú principal son:

(2) Declaración de variables para el menú principal del CONFYP

```
Var
  M1 MENU
  C,P,F,R,S POPUPMENU
  THECHOICE,OPCION STRING
  FORMACAT,MAINPAGE FORM
EndVar
```

(3) Método llamado para maximizar la pantalla
Method Arrive(Var EventInfo MoveEvent)
Maximize()
EndMethod

(4) Método utilizado seleccionar dentro del menú principal una opción deseada
Method MenuAction(Var EventInfo MenuEvent)
TheChoice=EventInfo.MenuChoice()
Switch
Case THECHOICE="&Dependencias" :
Message("La opción del Catálogo de Dependencias activada.")
OPCION="DEPENDEN"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Empresas" :
Message("La opción del Catálogo de Empresas activada.")
OPCION="EMPRESAS"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="Edo_&Facturas" :
Message("La opción del Catálogo de Estados de la Factura activada.")
OPCION="EDOFACT"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="Edo_&Proyectos" :
Message("La opción del Catálogo de Estados del Proyecto activada.")
OPCION="EDOPROY"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Gerencias" :
Message("La opción del Catálogo de Gerencias activada.")
OPCION="GERENCIA"
LLama_Opcion(OPCION)

CONTINUACION (4A)

```
Message("")
Case THECHOICE="&Responsables" :
Message("La opción del Catálogo de Responsables activada.")
OPCION="RESPONSA"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Subgerencias" :
Message("La opción del Catálogo de Subgerencias activada.")
OPCION="SUBGEREN"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Tipos"      :
Message("La opción del Catálogo de Tipos de Proyecto activada.")
OPCION="TIPOS"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Facturas"   :
Message("La opción de Facturas activada.")
OPCION="FACTURAS"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Proyectos"  :
Message("La opción de Proyectos activada.")
OPCION="PROYECTOS"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Reportes"   :
Message("La opción de Reportes activada.")
OPCION="SALIDARE"
LLama_Opcion(OPCION)
Message("")
Case THECHOICE="&Si"         :Salida()
```

CONTINUACION (4B)
Case THECHOICE="&No" ;;
EndSwitch
EndMethod

(5) Método que permite desplegar el menú y activarlo en la forma MENU1
Method Open(Var EventInfo Event)
HideSpeedBar()
Edit()
C.AddText("&Dependencias")
C.AddText("&Empresas")
C.AddText("Edo_&Facturas")
C.AddText("Edo_&Proyectos")
C.AddText("&Gerencias")
C.AddText("&Responsables")
C.AddText("&Subgerencias")
C.AddText("&Tipos")
S.AddText("&Si")
S.AddText("&No")
M1.AddPopUp("&Catálogos",C)
M1.AddPopUp("&Proyectos",P)
M1.AddPopUp("&Facturas",F)
M1.AddPopUp("&Reportes",R)
M1.AddPopUp("&Salida",S)
M1.Show()
Self.MoveTo()
EndMethod

(6) Procedimiento que asigna y llama a una forma desde una primera forma

```
Proc LLama_Opcion(OPCION STRING)
If IsAssigned(FORMACAT)
 Then
Maximize()
FORMACAT.BringtoTop()
FORMACAT.Wait()
FORMACAT.Hide()
MAINPAGE.MoveTo()
Else
 If FORMACAT.Open(OPCION+"FSL", WinStyleMaximize)
Then
   Maximize()
   FORMACAT.Moveto()
   FORMACAT.Wait()
   FORMACAT.Hide()
   MAINPAGE.MoveTo()
 EndIf
EndIf
EndProc
```

(7) Procedimiento que permite salir del menú principal del CONFYP

```
Proc Salida()
If MsgYesNoCancel("Salir de CONFYP","Esta seguro de querer abandonar el
Sistema?")="Yes" then
 RemoveMenu()
 Close()
EndIf
EndProc
```

(8) Método que permite accesar al catálogo de Dependencias de Pemex

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("DEPENDEN.FSL")
Endmethod
```

(9) Método que permite accesar al catálogo de Empresas

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("EMPRESAS.FSL")
EndMethod
```

(10) Método que permite accesar al catálogo de Gerencias de Pemex

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("GERENCIA.FSL")
EndMethod
```

(11) Método que permite accesar al catálogo de Estados de la Factura

```
Method pushButton(var eventInfo Event)
 FORMACAT.Open("EDOFACT.FSL")
endmethod
```

(12) Método que permite accesar al catálogo de Estados del proyecto

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("EDOPROY.FSL")
EndMethod
```

(13) Método que permite accesar al catálogo de Tipos de proyectos

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("TIPOS.FSL")
EndMethod
```

(14) Método que permite accesar al catálogo de Subgerencias de Pemex

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("SUBGEREN.FSL")
EndMethod
```

(15) Método que permite accesar al catálogo de Responsables de Pemex

```
Method PushButton(Var EventInfo Event)
 FORMACAT.Open("RESPONSA.FSL")
EndMethod
```

(16) Método que permite accesar a la opción de Proyectos

```
Method PushButton(Var EventInfo Event)
  FORMACAT.Open("PROYECTO.FSL")
EndMethod
```

(17) Método que permite accesar a la opción de Facturas

```
Method PushButton(Var EventInfo Event)
  FORMACAT.Open("FACTURAS.FSL")
EndMethod
```

3.- Entrada al módulo del catálogo de Empresas

Se toma como referencia estos programas fuentes para la ejecución de las opciones de todos los catálogos del sistema.

(18) Declaración de variables para la opción del catálogo de Empresas

```
Var
 CURSOR TCURSOR
 SITEFORM FORM
 M2 MENU
```

CONTINUACION (18)

```
CI,CE,CS,CR,CP POPUPMENU
THECHOICE STRING
REPINFO REPORTPRINTINFO
NCOPIES SMALLINT
R1REP REPORT
EndVar
```

(19) Método que especifica un mensaje al entrar a la forma del catálogo de Empresas

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
            ;
        Else
   Message("Bienvenidos al Catálogo de Empresas! Seleccione la opción deseada.")
EndIf
EndMethod
```

(20) Método que activa los errores de captura de la forma del catálogo de Empresas

```
Method Error(Var EventInfo ErrorEvent)
If EventInfo.IsPreFilter()
        Then
                ; This code executes for each object on the form.
  If EventInfo.ErrorCode()=9729 Then
   Beep()
   Sleep(100)
   MsgStop("Error en la captura.","El valor de la clave de la Empresa no puede ser duplicada.")
  EndIf
        Else
                ; This code executes only for the form.
EndIf
EndMethod
```

(21) Método utilizado para seleccionar dentro del menú del catálogo de Empresas la opción deseada

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
THECHOICE=EventInfo.MenuChoice()
Switch
Case THECHOICE="&Primero"   :
 Message("Primer registro")
 Active.Action(DataBegin)
 Message("")
Case THECHOICE="&Ultimo"    :
 Message("Ultimo registro")
 Active.Action(DataEnd)
 Message("")
Case THECHOICE="&Siguiente" :
 Message("Siguiente registro")
 Active.Action(DataNextRecord)
 Message("")
Case THECHOICE="&Anterior"  :
 Message("Anterior registro")
 Active.Action(DataPriorRecord)
 Message("")
Case THECHOICE="&Buscar"    :
 Message("Buscar registro")
 Active.Action(DataSearch)
 Message("")
Case THECHOICE="&Borrar"    :
 Message("Borrar registro")
 Active.Action(DataDeleteRecord)
 Active.Action(DataBegin)
 Message("")
Case THECHOICE="&Insertar"  :
```

CONTINUACION (21)

```
  Message("Insertar registro")
  Active.Action(DataInsertRecord)
  Message("")
 Case THECHOICE="&Modificar"  :
  Message("Modificar registro")
  Active.Action(DataBeginEdit)
  Message("")
 Case THECHOICE="&Imprimir"        : Salida_Impresora()
 Case THECHOICE="&Si"              : Salida()
 Case THECHOICE="&No"              : ;No realiza nada
EndSwitch
EndMethod
```

(22) Método que permite desplegar el menú y activarlo en la opción de Empresas

```
Method Open(Var EventInfo Event)
HideSpeedBar()
Edit()
CR.AddText("&Primero")
CR.AddText("&Ultimo")
CR.AddText("&Siguiente")
CR.AddText("&Anterior")
CR.AddText("&Buscar")
M2.AddPopUp("&Registro",CR)
CE.AddText("&Borrar")
CE.AddText("&Insertar")
CE.AddText("&Modificar")
M2.AddPopUp("&Edición",CE)
M2.AddPopUp("&Imprimir",CI)
CS.AddText("&Si")
CS.AddText("&No")
```

CONTINUACION (22)

```
M2.AddPopUp("&Salida",CS)
M2.Show()
EndMethod
```

(23) Procedimiento que permite accesar al reporte por impresora del reporte del catálogo de Empresas

```
Proc Salida_Impresora()
REPINFO.NCopies=1
REPINFO.MakeCopies=True
REPINFO.Name="REP_EMP"
RIREP.Print(RepInfo)
EndProc
```

(24) Procedimiento que permite salir de la opción del catálogo de Empresas

```
Proc Salida()
If MsgYesNoCancel("Salir del Catálogo de Empresas","Esta seguro de querer
abandonar este Catálogo de Empresas?")="Yes" Then
 FormReturn(True)
 Maximize()
EndIf
EndProc
```

(25) Método que activa la acción de posición del cursor en el primer registro del tabla de Empresas

```
Method PushButton(Var EventInfo Event)
   Active.Action(DataBegin)
EndMethod
```

(26) Método que activa la acción de posición del cursor en el anterior registro a partir del cual este posicionado

```
Method PushButton(Var EventInfo Event)
  Active.action(DataPriorRecord)
EndMethod
```

(27) Método que activa la acción de posición del cursor en el siguiente registro a partir del cual este posicionado en el tabla del catálogo de Empresas

```
Method PushButton(Var EventInfo Event)
  Active.action(DataNextRecord)
EndMethod
```

(28) Método que activa la acción de posición del cursor en el último registro del tabla de Empresas

```
Method PushButton(Var EventInfo Event)
  Active.Action(DataEnd)
EndMethod
```

(29) Método para borrar un registro del tabla de Empresas de Pemex

```
Method PushButton(Var EventInfo Event)
  Message("Borrar registro")
  Active.Action(DataDelcteRecord)
  Active.Action(DataBegin)
  Message("")
Endmethod
```

(30) Método que permite accesar a un procedimiento para la impresión del reporte del catálogo de Empresas

```
Method PushButton(Var.EventInfo Event)
  Salida_Impresora()
Endmethod
```

(31) Método que permite accesar a un procedimiento para salir de la opción de catálogo de Empresas

```
Method PushButton(Var EventInfo Event)
  Salida()
Endmethod
```

4.- Entrada al módulo de Proyectos

Este módulo de proyectos es quizá uno de los más importantes del sistema, debido que aquí es donde se permite al usuario capturar toda la información relacionada con los proyectos en proceso de desarrollo. Para tener un control de estos proyectos se generan opciones de (altas, bajas, modificaciones y consultas(pantalla e impresora). Todos estos procesos se ejecutan a partir de la forma de proyectos, como se observa a continuación en los siguientes programas fuentes.

(32) Método que permite el desplegado de un mensaje en el momento de abrir la forma de proyectos

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
          ;
        Else
   Message("Bienvenidos a PROYECTOS! Seleccione la opción deseada.")
Endif
EndMethod
```

(33) Método que activa los errores de la captura de proyectos en la forma proyectos

```
Method Error(Var EventInfo ErrorEvent)
If EventInfo.IsPreFilter()
        Then
                ; This code executes for each object on the form.
  If EventInfo.ErrorCode()<>0 Then
   Beep()
   Sleep(100)
   Switch
```

CONTINUACION (33)

```
    Case ErrorCode()=9732  :
      MsgStop("Error en la captura.","Este campo es requerido.")
    Case ErrorCode()=9736  :
      MsgStop("Error en la captura.","Este campo requiere de el llamado de un catálogo oprima
[CTRL-SPACE].")
    Case ErrorCode()=9729  :
      MsgStop("Error en la captura.","La clave no puede ser duplicada.")
    Case ErrorCode()=-31983 :
      MsgStop("Error en el formato de fecha.","El número de día no es válido.")
    Case ErrorCode()=-31984 :
      MsgStop("Error en el formato de fecha.","El número de mes no es válido.")
    Case ErrorCode()=-31738 :
      MsgStop("Error en el formato de fecha.","El formato de fecha dado es incorrecto.")
  EndSwitch
 EndIf
        Else
                ; This code executes only for the form.
EndIf
EndMethod
```

(34) Método utilizado para seleccionar dentro del menú de proyectos la opción deseada

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
THECHOICE=EventInfo.MenuChoice()
Switch
Case THECHOICE="&Primero"   :
  Message("Primer registro")
  Active.Action(DataBegin)
  Message("")
Case THECHOICE="&Ultimo"    :
  Message("Ultimo registro")
```

CONTINUACION (34)

```
 Active.Action(DataEnd)
 Message("")
Case THECHOICE="&Siguiente"  :
 Message("Siguiente registro")
 Active.Action(DataNextRecord)
 Message("")
Case THECHOICE="&Anterior"   :
 Message("Anterior registro")
 Active.Action(DataPriorRecord)
 Message("")
Case THECHOICE="&Buscar"     :
 Message("Buscar registro")
 Active.Action(DataSearch)
 Message("")
Case THECHOICE="&Borrar"     :
 Message("Borrar registro")
 Active.Action(DataDeleteRecord)
 Active.Action(DataBegin)
 Message("")
Case THECHOICE="&Insertar"   :
 Message("Insertar registro")
 Active.Action(DataInsertRecord)
 Message("")
Case THECHOICE="&Modificar"  :
 Message("Modificar registro")
 Active.Action(DataBeginEdit)
 Message("")
Case THECHOICE="&Pantalla"       : Salida_Pantalla()
Case THECHOICE="&Impresora"      : Salida_Impresora()
Case THECHOICE="&Si"             : Salida()
Case THECHOICE="&No"             : ¡No realiza nada
EndSwitch:EndMethod
```

(35) Método que permite desplegar el menú de la opción de proyectos y activarlo

```
Method Open(Var EventInfo Event)
  HideSpeedBar()
  Edit()
  CR.AddText("&Primero")
  CR.AddText("&Ultimo")
  CR.AddText("&Siguiente")
  CR.AddText("&Anterior")
  CR.AddText("&Buscar")
  M2.AddPopUp("&Registro",CR)
  CE.AddText("&Borrar")
  CE.AddText("&Insertar")
  CE.AddText("&Modificar")
  M2.AddPopUp("&Edición",CE)
  CI.AddText("&Impresora")
  CI.AddText("&Pantalla")
  M2.AddPopUp("&Consulta",CI)
  CS.AddText("&Si")
  CS.AddText("&No")
  M2.AddPopUp("&Salida",CS)
  M2.Show()
EndMethod
```

(36) Procedimiento que ejecuta un Query para generar una tabla de consulta de proyectos

```
Proc Genera_Consulta()
   ExecuteQBEFile("SALPRO01.QBE")
EndProc
```

(37) Procedimiento que permite obtener el reporte de proyectos por pantalla

```
Proc Salida_Pantalla()
  Genera_Consulta()
  SITEFORM.Open("SALPRO01.FSL")
```

CONTINUACION (38)

```
SITEFORM.Wait()
SiTTEFORM.Close()
EndProc
```

(38) Procedimiento que permite obtener el reporte de los proyectos por impresora

```
Proc Salida_Impresora()
 Genera_Consulta()
 REPINFO.NCopies=1
 REPINFO.MakeCopies=True
 REPINFO.Name="REP_PRO"
 RIREP.Print(RepInfo)
EndProc
```

(39) Procedimiento que permite salir de la opción de proyectos

```
Proc Salida()
 If MsgYesNoCancel("Salir del Proyectos","Esta seguro de querer abandonar
Proyectos?")="Yes" Then
  FormReturn()
  Maximize()
EndIf
EndProc
```

(40) Método que activa la acción de posición del cursor en el primer registro del tabla de proyectos

```
Method PushButton(Var EventInfo Event)
   Active.Action(DataBegin)
EndMethod
```

(41) Método que activa la acción de posición del cursor en el anterior registro a partir del cual este posicionado

```
Method PushButton(Var EventInfo Event)
  Active.action(DataPriorRecord)
EndMethod
```

(42) Método que activa la acción de posición del cursor en el siguiente registro a partir del cual este posicionado en el tabla de proyectos

```
Method PushButton(Var EventInfo Event)
  Active.action(DataNextRecord)
EndMethod
```

(43) Método que activa la acción de posición del cursor en el último registro del tabla de proyectos

```
Method PushButton(Var EventInfo Event)
  Active.Action(DataEnd)
EndMethod
```

(44) Método para borrar un registro del tabla de Empresas de Pemex

```
Method PushButton(Var EventInfo Event)
  Message("Borrar registro")
  Active.Action(DataDeleteRecord)
  Active.Action(DataBegin)
  Message("")
Endmethod
```

(45) Método que permite accesar a un procedimiento para la impresión del reporte del proyectos

```
Method PushButton(Var EventInfo Event)
  Salida_Impresora()
Endmethod
```

(46) Método que permite accesar a un procedimiento para salir de la opción proyectos

```
Method PushButton(Var EventInfo Event)
  Salida()
Endmethod
```

5.- Entrada al módulo de Facturas

Al igual que en el módulo de proyectos al ejecutarse la opción facturas se accesa a una forma principal de facturas la cual contiene los programas fuentes (métodos, procedimientos, declaración de variables, mensajes de error, querys, etc.) correspondientes para el manejo interno de la opción, como a continuación se presenta.

(47) Declaración de variables para la opción de facturas

```
Var
CURSOR TCURSOR
SITEFORM FORM
M2 MENU
CI,CE,CS,CR,CP POPUPMENU
THECHOICE STRING
REPINFO REPORTPRINTINFO
NCOPIES SMALLINT
RIREP REPORT
CONSUL QUERY
EndVar
```

(48) Método que permite desplegar un mensaje en el momento que se abre la forma de facturas

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
          ;
        Else
  Message("Bienvenidos a FACTURAS!  Seleccione la opción deseada.")
EndIf
EndMethod
```

(49) Método que activa los errores ocurridos en forma facturas en el momento de capturar información

```
Method Error(Var EventInfo ErrorEvent)
If EventInfo.IsPreFilter()
        Then
                ; This code executes for each object on the form.
  If EventInfo.ErrorCode()<>0 Then
   Beep()
   Sleep(100)
   Switch
    Case ErrorCode()=9732  :
     MsgStop("Error en la captura.","Este campo es requerido.")
    Case ErrorCode()=9736  :
     MsgStop("Error en la captura.","Este campo requiere de el llamado de un
catálogo oprima [CTRL-SPACE].")
    Case ErrorCode()=9729  :
     MsgStop("Error en la captura.","La clave no puede ser duplicada.")
    Case ErrorCode()=-31983 :
     MsgStop("Error en el formato de fecha.","El número de día no es
válido.")
    Case ErrorCode()=-31984 :
     MsgStop("Error en el formato de fecha.","El número de mes no es
válido.")
```

CONTINUACION (49)

```
    Case ErrorCode()=-31738 :
     MsgStop("Error en el formato de fecha.","El formato de fecha dado es
incorrecto.")
   EndSwitch
  EndIf
        Else
        ; This code executes only for the form.
EndIf
EndMethod
```

(50) Método utilizado para seleccionar dentro del menú del facturas la opción deseada

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
THECHOICE=EventInfo.MenuChoice()
Switch
 Case THECHOICE="&Primero"   :
  Message("Primer registro")
  Active.Action(DataBegin)
  Message("")
 Case THECHOICE="&Ultimo"    :
  Message("Ultimo registro")
  Active.Action(DataEnd)
  Message("")
 Case THECHOICE="&Siguiente" :
  Message("Siguiente registro")
  Active.Action(DataNextRecord)
  Message("")
 Case THECHOICE="&Anterior"  :
  Message("Anterior registro")
  Active.Action(DataPriorRecord)
  Message("")
 Case THECHOICE="&Buscar"    :
  Message("Buscar registro")
  Active.Action(DataSearch)
  Message("")
 Case THECHOICE="&Borrar"    :
  Message("Borrar registro")
  Active.Action(DataDeleteRecord)
  Active.Action(DataBegin)
  Message("")
 Case THECHOICE="&Insertar"  :
  Message("Insertar registro")
```

CONTINUACION (50)

```
  Active.Action(DataInsertRecord)
  Message("")
 Case THECHOICE="&Modificar"  :
  Message("Modificar registro")
  Active.Action(DataBeginEdit)
  Message("")
 Case THECHOICE="&Pantalla"       : Salida_Pantalla()
 Case THECHOICE="&Impresora"      : Salida_Impresora()
 Case THECHOICE="&Si"             :  Salida()
 Case THECHOICE="&No"             : ;No realiza nada
EndSwitch
EndMethod
```

(51) Método que permite desplegar el menú de facturas y activarlo

```
Method Open(Var EventInfo Event)
  HideSpeedBar()
  Edit()
  CR.AddText("&Primero")
  CR.AddText("&Ultimo")
  CR.AddText("&Siguiente")
  CR.AddText("&Anterior")
  CR.AddText("&Buscar")
  M2.AddPopUp("&Registro",CR)
  CE.AddText("&Borrar")
  CE.AddText("&Insertar")
  CE.AddText("&Modificar")
  M2.AddPopUp("&Edición",CE)
  CI.AddText("&Impresora")
  CI.AddText("&Pantalla")
  M2.AddPopUp("&Consulta",CI)
  CS.AddText("&Si")
  CS.AddText("&No")
  M2.AddPopUp("&Salida",CS):M2.Show():EndMethod
```

(52) Procedimiento que ejecuta un Query generando una tabla de consulta de facturas

```
Proc Genera_Consulta()
  ExecuteQBEFile("SALFAC01.QBE")
EndProc
```

(53) Procedimiento que permite generar un reporte de facturas por pantalla

```
Proc Salida_Pantalla()
 Genera_Consulta()
 SITEFORM.Open("SALFAC01.FSL")
 SITEFORM.Wait()
 SITEFORM.Close()
EndProc
```

(54) Procedimiento que permite generar el reporte de facturas por impresora

```
Proc Salida_Impresora()
 Genera_Consulta()
 REPINFO.NCopies=1
 REPINFO.MakeCopies=True
 REPINFO.Name="REP_FAC"
 R1REP.Print(RepInfo)
EndProc
```

(55) Procedimiento que coloca una bandera en el tabla de fechas de las facturas indicando el última Dependencia donde se encuentra la factura

```
Proc Genera_XFechas()
Var
 NOMTABLA STRING
 TABLA TABLE
 NUMREC NUMBER
EndVar
NOMTABLA="FECHFACT.DB"
```

CONTINUACION (55)

```
NUMREC=0
I=0
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  PROYECTO1=CURSOR.CVE_PROYECTO
  PROYECTO2=PROYECTO1
  While (I<=NUMREC)
    While (PROYECTO1=PROYECTO2) AND (I<=NUMREC)
      CURSOR.NextRecord()
      PROYECTO2=CURSOR.CVE_PROYECTO
      I=I+1
    EndWhile
    If (I-1=NUMREC) Then
      CURSOR.BANDERA="X"
    Else
      CURSOR.PriorRecord()
      CURSOR.BANDERA="X"
      CURSOR.NextRecord()
    EndIf
    PROYECTO1=CURSOR.CVE_PROYECTO
    PROYECTO2=PROYECTO1
  EndWhile
 EndIf
EndIf
EndProc
```

(56) Procedimieto que permite salir de la opción de facturas

```
Proc Salida()
If MsgYesNoCancel("Salir del Facturas","Esta seguro de querer abandonar
Facturas?")="Yes" Then
 Genera_XFechas()
 FormReturn()
 Maximize()
EndIf EndProc
```

NOTA: En el método de PushButton de la opción de facturas efectúan la misma acción que los de proyectos por ello no se incluyen en esta opción de facturas.

6.- Entrada al módulo de Reportes

Sin duda el objetivo de todo sistema de software de este tipo, es las salidas que se puedan obtener a partir de él. Todas estas salidas con referencias a la información capturada por el usuario.

(57) Declaración de variables para la opción de reportes

```
Var
 CURSOR TCURSOR
 SITEFORM FORM
 M2 MENU
 CI,CG,CS POPUPMENU
 THECHOICE STRING
 REPINFO REPORTPRINTJNFO
 NCOPIES SMALLINT
 R1REP REPORT
 REPORTE STRING
 INCORR NUMBER
 NUMREC NUMBER
 TABLA TABLE
 NOMTABLA STRING
EndVar
```

(58) Método que permite desplegar un mensaje en el momento que entra a la opción de reportes del CONFYP

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
        ;
        Else
   Message("Bienvenidos a los reportes del  C O N F Y P.!")
EndIf
EndMethod
```

(59) Método que permite seleccionar una opción dentro del menú de reportes

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
THECHOICE=EventInfo.MenuChoice()
Switch
 Case THECHOICE="(&G1) Control financiero p/c proyecto"  :
  Message("CONTROL FINANCIERO POR PROYECTO GRAFICO")
  REPORTE="SALGRA01"
  Salida_Pantalla(REPORTE)
  Message("")
 Case THECHOICE="(&R1) Control financiero p/c proyecto"  :
  Message("CONTROL FINANCIERO POR PROYECTO")
  REPORTE="SALCON01"
  Salida_Pantalla(REPORTE)
  Message("")
 Case THECHOICE="(&R2) Control financiero p/c factura"  :
  Message("CONTROL FINANCIERO POR FACTURA")
  REPORTE="SALCON02"
  Salida_Pantalla(REPORTE)
  Message("")
 Case THECHOICE="(&R3) Control de facturas"  :
  Message("CONTROL DE FACTURAS")
```

CONTINUACION (59)

```
  REPORTE="SALCON03"
  Salida_Pantalla(REPORTE)
  Message("")
Case THECHOICE="(&R4) Estado actual de facturas"     :
  Message("ESTADO ACTUAL DE FACTURAS")
  REPORTE="SALCON04"
  Salida_Pantalla(REPORTE)
  Message("")
Case THECHOICE="(&R5) Estado financiero de proyectos"  :
  Message("ESTADO FINANCIERO DE PROYECTOS")
  REPORTE="SALCON05"
  Salida_Pantalla(REPORTE)
  Message("")
Case THECHOICE="&Si"          : Salida()
Case THECHOICE="&No"          : ;No realiza nada
EndSwitch
EndMethod
```

(60) Método que despliega el menú de reportes y lo activa

```
Method Open(Var EventInfo Event)
 HideSpeedBar()
 Edit()
 CG.AddText("(&G1) Control financiero p/c proyecto")
 M2.AddPopUp("&Gráficas",CG)
 CI.AddText("(&R1) Control financiero p/c proyecto")
 CI.AddText("(&R2) Control financiero p/c factura")
 CI.AddText("(&R3) Control de facturas")
 CI.AddText("(&R4) Estado actual de facturas")
 CI.AddText("(&R5) Estado financiero de proyectos")
 M2.AddPopUp("&Reportes",CI)
 CS.AddText("&Si")
 CS.AddText("&No")
 M2.AddPopUp("&Salida",CS):M2.Show():EndMethod
```

(61) Procedimiento que verifica que el tabla de fechas de las facturas no este vacío

```
Proc Verifica_FechasFacturas(VAR INCORR NUMBER)
 NOMTABLA="FECHFACT.DB"
 NUMREC=0
 If IsTable(NOMTABLA) Then
  TABLA.Attach(NOMTABLA)
  NUMREC=TABLA.Nrecords()
  If NUMREC=0 Then
   INCORR=0
   MsgInfo("Error al generar el reporte.","La tabla de FECHAS de las facturas no tiene
registros.")
  Else
   INCORR=1
  EndIf
 EndIf
EndProc
```

(62) Procedimiento que verifica que el tabla de proyectos no este vacío

```
Proc Verifica_Proyectos(VAR INCORR NUMBER)
 NOMTABLA="PROYECTO.DB"
 NUMREC=0
 If IsTable(NOMTABLA) Then
  TABLA.Attach(NOMTABLA)
  NUMREC=TABLA.Nrecords()
  If NUMREC=0 Then
   INCORR=0
   MsgInfo("Error al generar el reporte.","La tabla de PROYECTOS no tiene registros.")
  Else
   INCORR=1
  EndIf
 EndIf
EndProc
```

(63) Procedimiento que verifica que el tabla de facturas no se encuentre vacío

```
Proc Verifica_Facturas(VAR INCORR NUMBER)
 NOMTABLA="FACTURAS.DB"
 NUMREC=0
 If IsTable(NOMTABLA) Then
  TABLA.Attach(NOMTABLA)
  NUMREC=TABLA.Nrecords()
  If NUMREC=0 Then
   INCORR=0
   MsgInfo("Error al generar el reporte.","La tabla de FACTURAS no tiene registros.")
  Else
   INCORR=1
  EndIf
 EndIf
EndProc
```

(64) Procedimiento que verifica que el tabla de presupuestos no este vacío

```
Proc Verifica_Presupuestos(VAR INCORR NUMBER)
 NOMTABLA="PRESUPUE.DB"
 NUMREC=0
 If IsTable(NOMTABLA) Then
  TABLA.Attach(NOMTABLA)
  NUMREC=TABLA.Nrecords()
  If NUMREC=0 Then
   INCORR=0
   MsgInfo("Error al generar el reporte.","La tabla de PRESUPUESTOS no tiene registros.")
  Else
   INCORR=1
  EndIf
 EndIf
EndProc
```

(65) Procedimiento que válida las tablas de necesarias para generar el reporte 1

```
Proc Valida_Sal01(VAR INCORR NUMBER)
NOMTABLA="SALIDARE.DB"
NUMREC=0
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  If (CURSOR."AÑO">0) And (CURSOR."CVE_PROYECTO"<>"") And
(CURSOR.FECHA_INICIAL<>"") And (CURSOR.FECHA_FINAL<>"") Then
   INCORR=1
   Verifica_Facturas(INCORR)
   If INCORR=1 Then
    Verifica_Presupuestos(INCORR)
   EndIf
  Else
   INCORR=0
   MsgInfo("Error en los parámetros de entrada.","El Año, la Fecha de inicio, la Fecha de
término y la clave del Proyecto son datos requeridos para la generación de este reporte.")
  EndIf
 EndIf
EndIf
EndProc
```

(66) Procedimiento que válida las tablas de necesarias para generar el reporte 2

```
Proc Valida_Sal02(VAR INCORR NUMBER)
NOMTABLA="SALIDARE.DB"
NUMREC=0
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC<>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  If (CURSOR."CVE_FACTURA"<>"")Then
   INCORR=1
   Verifica_Facturas(INCORR)
   If (INCORR=1) Then
    Verifica_FechasFacturas(INCORR)
   EndIf
  Else
   INCORR=0
   MsgInfo("Error en los parámetros de entrada.","La clave de la Factura es requerida para la
generación de este reporte.")
  EndIf
 EndIf
EndIf
EndProc
```

(67) Procedimiento que válida las tablas de necesarias para generar el reporte3

```
Proc Valida_Sal03(VAR INCORR NUMBER)
NOMTABLA="SALIDARE.DB"
NUMREC=0
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC<>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  If (CURSOR.FECHA_INICIAL<>"") And (CURSOR.FECHA_FINAL<>"") Then
   INCORR=1
    Verifica_Facturas(INCORR)
  Else
   INCORR=0
   MsgInfo("Error en los parámetros de entrada.","La Fecha de inicio, la Fecha de término y la
clave del Proyecto son datos requeridos para la generación de este reporte.")
  EndIf
 EndIf
EndIf
EndProc
```

(68) Procedimiento que elige un reporte u otro dependiendo de las características de los parámetros asignados

```
Proc Elegir_Reporte(VDEPENDENCIA STRING,VAR REPORTE STRING)
Switch
 Case VDEPENDENCIA="" : REPORTE="SALCON4A"
 Case VDEPENDENCIA<>"" : REPORTE="SALCON4B"
EndSwitch
EndProc
```

(69) Procedimiento que elige un reporte u otros dependiendo de las características de los parámetros asignados

```
Proc Elegir_Reporte1(VGERENCIA STRING,VSUBGERENCIA STRING,VEMPRESA STRING,VAR
REPORTE STRING,VAR BAND NUMBER)
Switch
 Case VGERENCIA=" AND VSUBGERENCIA="" AND VEMPRESA="" : REPORTE="SALCON5A"
 Case VGERENCIA<>"" AND VSUBGERENCIA<>"" AND VEMPRESA<>"" :
  REPORTE="SALCON5D"
  BAND=0
 Case VGERENCIA<>"" AND VSUBGERENCIA="" AND VEMPRESA="" :
  REPORTE="SALCON5B"
 Case VGERENCIA="" AND VSUBGERENCIA<>"" AND VEMPRESA="" :
  REPORTE="SALCON5E"
 Case VGERENCIA="" AND VSUBGERENCIA="" AND VEMPRESA<>"" :
  REPORTE="SALCON5G"
 Case VGERENCIA<>"" AND VSUBGERENCIA<>"" AND VEMPRESA="" :
  REPORTE="SALCON5C"
  BAND=0
 Case VGERENCIA="" AND VSUBGERENCIA<>"" AND VEMPRESA<>"" :
  REPORTE="SALCON5F"
  BAND=0
 Case VGERENCIA<>"" AND VSUBGERENCIA="" AND VEMPRESA<>"" :
  REPORTE="SALCON5H"
  BAND=0
EndSwitch
EndProc
```

(70) Procedimiento que válida las tablas de necesarias para generar el reporte 4

```
Proc Valida_Sal04(VAR INCORR NUMBER,VAR REPORTE STRING)
Var
 VDEPENDENCIA STRING
EndVar
NOMTABLA="SALIDARE.DB"
NUMREC=0
VDEPENDENCIA=""
```

CONTINUACION (70)

```
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  If (CURSOR.FECHA_INICIAL<>"") And (CURSOR.FECHA_FINAL<>"") Then
   VDEPENDENCIA=CURSOR.CVE_DEPENDENCIA
   Elegir_Reporte(VDEPENDENCIA,REPORTE)
   INCORR=1
   Verifica_Facturas(INCORR)
   If INCORR=1 Then
    Verifica_FechasFacturas(INCORR)
   EndIf
  Else
   INCORR=0
   MsgInfo("Error en los parámetros de entrada.","La Fecha de inicio, la Fecha de término y la clave del
Proyecto son datos requeridos para la generación de este reporte.")
  EndIf
 EndIf
EndIf
EndProc
```

(71) Procedimiento que válida las tablas de necesarias para generar el reporte 5

```
Proc Valida_Sal05(VAR INCORR NUMBER,VAR REPORTE STRING,VAR BAND NUMBER)
Var
 VGERENCIA,VSUBGERENCIA,VEMPRESA STRING
EndVar
NOMTABLA="SALIDARE.DB"
NUMREC=0
VGERENCIA=""
VSUBGERENCIA=""
VEMPRESA=""
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 If NUMREC>0 Then
  CURSOR.Open(NOMTABLA)
  CURSOR.Home()
  CURSOR.Edit()
  If (CURSOR.AÑO<>"") AND (CURSOR.FECHA_INICIAL<>"") And (CURSOR.FECHA_FINAL<>"") Then
   VGERENCIA=CURSOR.CVE_GERENCIA
   VSUBGERENCIA=CURSOR.CVE_SUBGERENCIA
   VEMPRESA=CURSOR.CVE_EMPRESA
   Elegir_Reporte1(VGERENCIA,VSUBGERENCIA,VEMPRESA,REPORTE,BAND)
   INCORR=1
   Verifica_Proyectos(INCORR)
   If INCORR=1 Then
    Verifica_Facturas(INCORR)
   Endif
   If INCORR=1 Then
    Verifica_Presupuestos(INCORR)
   Endif
  Else
   INCORR=0
   MsgInfo("Error en los parámetros de entrada.","La Fecha de inicio, la Fecha de término y la clave del Proyecto
son datos requeridos para la generación de este reporte.")
  Endif
```

CONTINUACION (71)

```
 Endif
Endif
EndProc
```

(72) Procedimiento que verifica la obtención de los reportes

```
Proc Verificar_SALIDAREP(VAR INCORR NUMBER,REPORTE STRING)
 NOMTABLA=REPORTE+".DB"
 NUMREC=0
 If IsTable(NOMTABLA) Then
  TABLA.Attach(NOMTABLA)
  NUMREC=TABLA.Nrecords()
  If NUMREC=0 Then
   INCORR=0
   MsgInfo("Error al generar el reporte.","La tabla de SALIDA para generar el reporte no tiene
registros.")
  Else
   INCORR=1
  Endif
 Endif
EndProc
```

(73) Procedimiento que ejecuta el reporte y lo despliega por pantalla o por impresora según las características del reporte

```
Proc Salida_Pantalla(REPORTE STRING)
Var
 NOMBREQ,NOMBREF STRING
 BAND NUMBER
EndVar
INCORR=0
BAND=1
Switch
  Case REPORTE="SALCON01": Valida_Sal01(INCORR)
  Case REPORTE="SALGRA01": Valida_Sal01(INCORR)
  Case REPORTE="SALCON02": Valida_Sal02(INCORR)
  Case REPORTE="SALCON03": Valida_Sal03(INCORR)
  Case REPORTE="SALCON04": Valida_Sal04(INCORR,REPORTE)
  Case REPORTE="SALCON05": Valida_Sal05(INCORR,REPORTE,BAND)
EndSwitch
If INCORR=1 Then
  NOMBREQ=REPORTE+".QBE"
  NOMBREF=REPORTE+".FSL"
  ExecuteQBEFile(NOMBREQ)
  Verificar_SALIDAREP(INCORR,REPORTE)
  If (INCORR=1) AND (BAND=1) Then
    SITEFORM.Open(NOMBREF)
    SITEFORM.Wait()
    SITEFORM.Close()
  Else
    If (INCORR=1) Then
      REPINFO.NCopies=1
      REPINFO.MakeCopies=True
      REPINFO.Name=REPORTE
      R1REP.Print(REPINFO)
    EndIf
  EndIf
Else
```

CONTINUACION (74)

```
MsgInfo("Imposible poder generar el reporte.","No existe información suficiente para la generación del mismo. Verificar tablas y parámetros de entrada.")
Endif
EndProc
```

(74) Procedimiento que borra los registros del tabla de salida

```
Proc Borrar_Registro()
Var
 I NUMBER
EndVar
NUMREC=0
I=1
NOMTABLA="SALIDARE.DB"
If IsTable(NOMTABLA) Then
 TABLA.Attach(NOMTABLA)
 NUMREC=TABLA.Nrecords()
 Cursor.Open(NOMTABLA)
 Cursor.Home()
 Cursor.Edit()
 If NUMREC>0 Then
  While I<=NUMREC
   Cursor.DeleteRecord()
   Cursor.NextRecord()
   I=I+1
  EndWhile
 Endif
Endif
EndProc
```

(75) Procedimiento que permite salir de la opción de reportes

```
Proc Salida()
If MsgYesNoCancel("Salir de Reportes de Salida","Esta seguro de querer
abandonar esta opción?")="Yes" Then
 Borrar_Registro()
 FormReturn()
 Maximize()
Endif
EndProc
```

REPORTE
CONTROL FINANCIERO POR PROYECTO

(76) Declaración de variables para la generación del reporte

```
Var
 M2 MENU
 CI,CE,CS,CR,CP POPUPMENU
 THECHOICE STRING
 REPINFO REPORTPRINTINFO
 NCOPIES SMALLINT
 R1REP REPORT
 REPORTE STRING
EndVar
```

(77) Método que permite desplegar un mensaje en el momento de entrar al reporte

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
                ;
        Else
  Message("Bienvenidos al reporte de Control financiero de proyectos!")
Endif
EndMethod
```

(78) Método que permite seleccionar dentro del menú del reporte la opción deseada

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
TheChoice=EventInfo.MenuChoice()
Switch
 Case TheChoice="&Imprimir"        :
  Message("CONTROL FINANCIERO DE LOS PROYECTOS DE GDT")
  Salida_Impresora()
  Message("")
```

CONTINUACION (78)

```
 Case TheChoice="&Si"          : Salida()
 Case TheChoice="&No"          : ;No realiza nada
EndSwitch
EndMethod
```

(79) Médodo que permite desplegar y activar el menú del reporte

```
Method Open(Var EventInfo Event)
 HideSpeedBar()
 Edit()
 M2.AddPopUp("&Imprimir",CI)
 CS.AddText("&Si")
 CS.AddText("&No")
 M2.AddPopUp("&Salida",CS)
 M2.Show()
EndMethod
```

(80) Procedimiento que permite obtener el reporte por pantalla

```
Proc Salida_Impresora()
RepInfo.NCopies=1
RepInfo.MakeCopies=True
RepInfo.Name="SALCON01"
R1Rep.Print(RepInfo):EndProc
```

(81) Procedimiento que permite salir de la opción reporte

```
Proc Salida()
If MsgYesNoCancel("Salir de este Reporte","Esta seguro de querer
abandonar esta opción?")="Yes" Then
 FormReturn(True)
 Maximize()
EndIf
EndProc
```

REPORTE
CONTROL FINANCIERO POR FACTURA

(82) Método que permite desplegar un mensaje en el momento que se abre la forma de este reporte

```
Method Arrive(Var EventInfo MoveEvent)
If EventInfo.IsPreFilter()
        Then
                ;
        Else
   Message("Bienvenidos al reporte de Control financiero de facturas!")
EndIf
EndMethod
```

(83) Método que permite seleccionar una opción dentro del menú de este reporte

```
Method MenuAction(Var EventInfo MenuEvent)
Active.Action(DataBegin)
Active.Action(DataBeginEdit)
THECHOICE=EventInfo.MenuChoice()
Switch
 Case THECHOICE="&Imprimir"      :
  Message("CONTROL FINANCIERO DE LAS FACTURAS DE GDT")
  Salida_Impresora()
  Message("")
```

Continuación del Método 83

```
Case THECHOICE="&Si"          : Salida()
Case THECHOICE="&No"          : ;No realiza nada
EndSwitch
EndMethod
```

(84) Método que despliega y activa el menú de el reporte

```
Method Open(Var EventInfo Event)
 HideSpeedBar()
 Edit()
 M2.AddPopUp("&Imprimir",CI)
 CS.AddText("&Si")
 CS.AddText("&No")
 M2.AddPopUp("&Salida",CS)
 M2.Show()
EndMethod
```

(85) Declaración de variables para la generación del reporte

```
Var
 M2 MENU
 CI,CE,CS,CR,CP POPUPMENU
 THECHOICE STRING
 RepInfo REPORTPRINTINFO
 NCOPIES SMALLINT
 R1REP REPORT
 REPORTE STRING
EndVar
```

(86) Procedimiento que permite generar el reporte por impresora

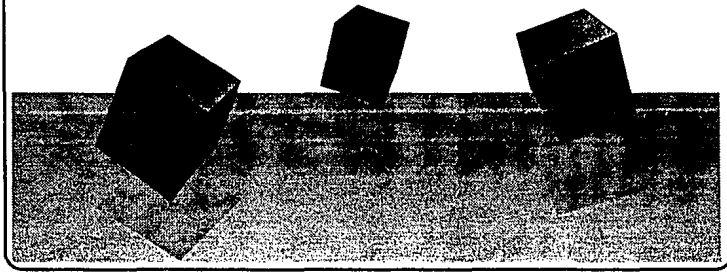
```
Proc Salida_Impresora()
 REPINFO.NCopies=1
 REPINFO.MakeCopies=True
 REPINFO.Name="SALCON02"
 R1REP.Print(RepInfo)
EndProc
```

(87) Procedimiento que accesa la salida del reporte

```
Proc Salida()
If MsgYesNoCancel("Salir de este Reporte","Esta seguro de querer abandonar
esta opción?")="Yes" Then
 FormReturn(True)
 Maximize()
EndIf
EndProc
```

CONCLUSIONES

VENTAJAS
DESVENTAJAS
PANORAMICA
ORIENTACION A OBJETOS



CONCLUSIONES

El mundo de la computación ha ido evolucionando aceleradamente tanto en la rama de software como en la de hardware. A pesar de esta evolución todavía no existe un software que cumpla eficazmente todos los requerimientos de analistas, diseñadores y programadores.

Debido a las necesidades de programadores por realizar software que cumpla con las exigencias de los empresarios, como son adquirir sistemas más productivos y a bajo costo, los diseñadores de software se ven en la necesidad de diseñar productos que resuelvan estos problemas. Es por esta razón que el mercado del software se ve inundado de productos día a día. Unas de las ideologías que actualmente se usa tanto en libros y revistas de computación así como de software es la llamada **Orientación a Objetos**. Cuyo tema ha interesado a investigadores para el desarrollo de software proponiendo nuevas metodologías para el desarrollo del mismo, con la finalidad de mejorarlo paulatinamente.

Uno de los autores más conocidos que se ha adentrado a este tema es **Grady Booch**, que crea una nueva metodología de análisis y diseño encaminada precisamente a mejorar el software; y establecer ciertos conceptos y criterios para aplicarla. A esta nueva visión de crear o desarrollar software se le denomina **Orientación a Objetos**.

Tomando en cuenta lo anterior podemos establecer que para un desarrollo de software orientado a objetos se puede considerar un lenguaje o una base de datos orientada a objetos como apoyo para el desarrollo del mismo. Actualmente existen varios lenguajes de programación orientados a objetos con diferentes características que los distinguen unos de otros, siendo unos más adecuados para determinado software que otros; además de lenguajes de programación como ya se mencionó anteriormente, las bases de datos orientadas a objetos están ocupando un papel muy importante en esta rama, dejando como opción también la utilización de estas para desarrollo de software; es importante visualizar que no podemos decir si un lenguaje es mejor que un

manejador de base de datos ya que esto es relativo y generalmente la utilización de uno u otro depende en gran medida del tipo de aplicación requerida.

En este documento se trato de visualizar, analizar y determinar relativamente todo lo que implica el desarrollo de una aplicación de software orientada a objetos, es decir, comenzar desde un análisis y diseño hasta llegar a la programación y terminación de la aplicación. Determinando con ello sus posibles ventajas y desventajas en relación a lo hecho tradicionalmente hasta el momento. Para establecer esto último fue necesario tomar una aplicación, realizar su análisis y diseño orientado a objetos utilizando el método de **Grady Booch**, elegir una base de datos orientada a objetos con determinadas características (**PARADOX for Windows versión 1.0**), y desarrollar el software.

Al terminar el desarrollo de la aplicación se estableció que utilizando un manejador de base de datos orientada a objetos en este caso (**PARADOX for Windows**), fue hasta cierto punto más rápido, que con un lenguaje de tercera generación como es el (**PASCAL versión 6.0**), provocando con ello un costo menor en la realización de la aplicación.

Existen infinidad de puntos que se tienen que analizar para determinar si esta ideología es mejor que las tradicionales, aunque actualmente existe mucho software orientado a objetos, todavía no se promueve esta innovación lo suficiente para tomarla de lleno.

De acuerdo a los fundamentos, y al desarrollo propiamente de una aplicación consideramos que la **orientación a objetos** podría definirse como **una ideología diferente de interpretar todos los aspectos de software; interviniendo en el análisis, diseño, programación,** lenguajes, base de datos, etc. Y pudiendo además considerar ciertas ventajas y desventajas como:

VENTAJAS

1.-Una nueva visión de interpretar el software

Esta nueva forma de interpretar el software facilita en gran medida las fases de análisis y diseño así como la evolución de una aplicación.

2.-Costo de software

La disminución del costo de desarrollo de software es posiblemente una ventaja de esta ideología, aunque se considera a la larga porque en un principio podría entenderse que es mucho más caro manejar un análisis, diseño y programación orientada a objetos, debido a lo que esto implica, como es la compra de software orientado a objetos, hardware, estudios sobre los fundamentos de la orientación a objetos y dependencia de terceras personas con más experiencia en el campo.

3.-Software reutilizable

En un mundo cambiante como es el nuestro donde siempre ocurre la necesidad de realizar modificaciones a nuestros sistemas de software de acuerdo a nuevos requerimientos, cabe la necesidad de tener una opción de procurar realizar un software modificable y reutilizable.

4.-Creación de sistemas de software con una alta calidad

Generalmente hace apenas unos años, el software que se realizaba era austero y poco amigable tanto para el programador como para el usuario del mismo. Ahora con esta perspectiva, cambia notablemente la calidad de un sistema de acuerdo a su características.

En conclusión podemos decir que los métodos orientados a objetos facilitan el diseño de bibliotecas de componentes reusables, las cuales una vez bien definidas, permitirán avanzar en la construcción de nuevos sistemas más rápidamente. La complejidad de sistemas se vuelve más manejable gracias a la posibilidad de jerarquizar las abstracciones de un sistema mediante las herencias entre clases, el trabajo de implementación se puede realizar fácilmente entre grupo de programadores, gracias a la poca y bien definida dependencia de las clases.

DESVENTAJAS

1.-Poco conocimiento de la ideología

Toda innovación requiere de un proceso de verificación y aceptación para su establecimiento, para ello es necesario un conocimiento profundo de los conceptos más elementales.

2.-Costo

Las herramientas como los lenguajes y manejadores de bases de datos orientadas a objetos que actualmente se manejan suelen tener un costo bastante elevado, así como los cursos que se imparte sobre este tema.

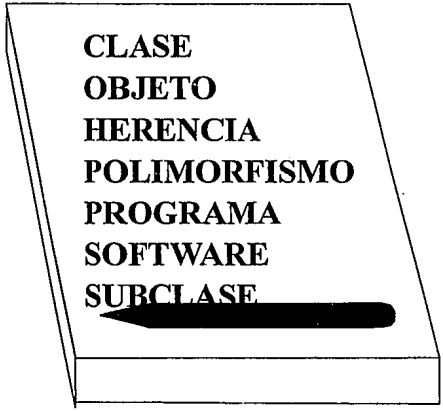
Todos los aspectos antes mencionados nos dejan ver claramente que no es tan absurdo querer propiciar de alguna forma un cambio de ideología de software ya que esto podría mejorar muchos elementos de software. Implicando con ello que esta visión pueda sucumbir el mundo de software en un tiempo no muy lejano.

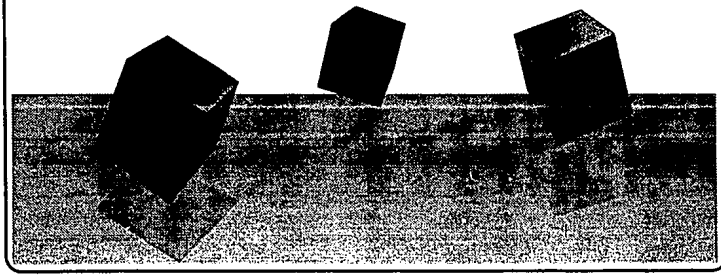
En general este documento proporciona al lector un conocimiento general de la visión de la orientación a objetos. Debido a que este tema es bastante amplió y incluye muchos factores que se tienen que estudiar profundamente sólo se trato de abarcar los conceptos generales.

Podemos concluir que los objetivos de este documento se cumplieron, debido a que se trato de abarcar todos los conceptos de la orientación a objetos incluyendo aspectos de programación, lenguajes, bases de datos, análisis y diseño (**Grady Booch**), además de profundizar en el análisis y diseño propuesto por el autor **Grady Booch** y hacer un breve análisis comparativo de esta metodología con la metodología clásica. Y considerar toda la teoría de esta nueva visión para aplicarla en un sistema de software.

Uno de los objetivos que se trato también de abarcar en este documento y que quizá no esta tan implícito en él como los demás, es hacer conciencia al lector de este tema y no pasarlo inadvertido, viendo claramente las ventajas y desventajas para el mundo del software. Por ello que se pretende dar al lector ciertos criterios para considerar esta metodología de orientación a objetos para que en un futuro se tome como otra alternativa el conjunto de aspectos de software (lenguajes, bases de datos, programación, análisis, diseño, etc.) orientadas a objetos para la generación de sistemas computacionales, evitando con ello desventajas que actualmente existen (complejidad, mantenimiento, costo, falta de reutilización, improductividad del programador, inflexibilidad para el cambio).

GLOSARIO


CLASE
OBJETO
HERENCIA
POLIMORFISMO
PROGRAMA
SOFTWARE
SUBCLASE



GLOSARIO

ABSTRACCION: Es una forma de caracterizar a los objetos fijándose solamente en propiedades que nos interesan en un momento dado e ignorando otros detalles que no son importantes.

ALTA: Permite añadir un registro dentro de los archivos.

ARCHIVO: Es la forma organizada de almacenar un grupo de registros con una función específica.

ANALISIS DE SISTEMAS: Son las tres primeras etapas del ciclo de desarrollo de sistemas, que comprenden el estudio de sistemas, la propuesta y el estudio de viabilidad; es el estudio formal de los objetivos del sistema, estructuras, flujos de información, grados de rendimiento y eficiencias.

ANALISIS ESTRUCTURADO: Es un planteamiento organizado para el análisis de sistemas que utilizan conjuntamente diagramas funcionales, análisis jerárquicos de la actividad empresarial y diagramas de flujo de datos.

ANALISIS ORIENTADO A OBJETOS: El análisis orientado a objetos es el que se encarga de descubrir los objetos semánticos, es decir las abstracciones que representan conceptos que tienen un significado claro en la descripción del problema.

BASE DE DATOS: Un conjunto de archivos de datos relacionados entre si.

BORRAR: Es borrar un registro de una tabla.

CAMPO: Número de caracteres que se pueden considerar en un dato.

CARACTER: Es cualquiera de las letras y dígitos representados en el teclado.

CLASE: Una clase es una colección de objetos similares. Es un prototipo que define los métodos y variables que serán incluidas en un tipo de objeto particular. La descripción de los métodos y variables que los soportan se describen sólo una vez, en la definición de la clase.

CLAVE: Es un campo que distingue de manera única un registro.

COMANDO: Es un conjunto de programas especiales para realizar un fin determinado.

COMPONENTE: Elemento primario, físico o conceptual, que satisface alguna función necesaria para un sistema para conseguir sus objetivos y que es controlado por el sistema.

CONSULTAR: Presenta en la pantalla los datos más importantes de Proyectos, Facturas.

CURSOR: Es una posición en un lugar dentro de la pantalla de la computadora que indica la espera de una tecla.

DIAGRAMA DE FLUJO: Método gráfico para describir algoritmos. En principio, todo algoritmo puede representarse mediante un diagrama de flujo, pero en la práctica este acercamiento tiene muchas desventajas, mismas que se han conducido a su casi total abandono, en favor de la técnica del "seudocódigo".

DIRECTORIO: Un directorio es un índice del contenido de un disco. Este contiene los nombres de los archivos, sus tamaños y las fechas en que fueron modificados por última vez.

DISEÑO ESTRUCTURADO: Conjunto de métodos para construir sistemas de información o, en general, para resolver problemas por medio de la computadora. El diseño estructurado parte de la base de que los programas se escriben estructuradamente, pero no se limita a consideraciones locales al programa, sino que estudia las relaciones entre todos los programas que constituyen un sistema.

DISEÑO ORIENTADO A OBJETOS: Es aquel que crea una representación del dominio del problema en el mundo real y lo transforma en un dominio de solución que es el software.
Construcción de sistemas de software como una colección estructurada de clases o implementaciones de tipo de datos abstractos.

ENCAPSULACION: Es la manera de ocultar los detalles de la representación interna de un objeto presentando solamente la interfaz disponible al usuario.

ENTER: Es una tecla que confirma la aceptación de un comando y/o dato.

HARDWARE: Conjunto de elementos y sistemas electrónicos que forman un sistema de cómputo. Dispositivo de una computadora o parte física.

HERENCIA: La relación de herencia impone una estructura jerárquica al conjunto de clases de un sistema. Bajo la relación de herencia, una clase puede heredar propiedades de otra clase.

INFORMATICA: Conjunto de conocimientos, métodos y sistemas para el manejo "computarizado" de la información en las organizaciones.

INGENIERIA DE SOFTWARE: Es una disciplina para desarrollar software de alta calidad para sistemas basados en computadora.

INSTANCIAS DE LA CLASE: Se le denomina de esta forma a los objetos que pertenecen a una clase y contienen tan solo los valores particulares para las variables, compartiendo el código de los métodos.

JERARQUIA: Es la manera de organizar y ordenar a las abstracciones.

MAQUINA DE TURIN : Modelo matemático de un autómata general. diseñado por Alan Turin y se emplea como medio de análisis en el trabajo teórico, y como herramienta para probar teoremas y proposiciones en matemáticas computacionales y teoría del lenguaje.

MENSAJE: La forma en que los objetos interactúan unos con otros es enviándose mensajes pidiendo que se ejecute un método específico. Un mensaje consiste simplemente del nombre del objeto a quien va dirigido seguido del nombre del método que el objeto receptor sabe como ejecutar. Si el método requiere información adicional, el mensaje incluye esa información como parámetros.

METODO: (Sinónimos: procedimientos, funciones, servicios). Son las operaciones que son capaces de realizar los objetos de una clase.

Cuando hay herencia de clases, los métodos definidos en una superclase se heredan a los objetos de las subclases.

MODULARIDAD: Un módulo es un programa que realiza una función específica, posee un método de solución para una tarea que se le ha sido encomendada así como el conocimiento suficiente para realizar ese trabajo.
La descomposición de un sistema en módulos debe efectuarse bajo una metodología que facilite la producción de componentes que puedan ser fácilmente combinados con otros, bajo reglas específicas para producir un sistema.

LENGUAJE DE ALTO NIVEL: Son los lenguajes de programación que están "por encima" (en poder expresivo) del lenguaje máquina y lenguaje ensamblador; pero para cada uno de ellos debe existir un compilador que traduzca el programa en él lenguaje de la máquina donde se intente ejecutar.

LENGUAJE DE MAQUINA: La única manera de comunicarse con el procesador de una computadora es por medio de un programa directamente ejecutable, mismo que debe estar escrito forzosamente en este lenguaje. Sin embargo el lenguaje de máquina no es propiamente un lenguaje, porque carece de estructura; por esta razón podría describirse como "conjunto de signos aislados ejecutables".

LENGUAJE DE PROGRAMACION: Nombre genérico que se aplica a cualquier lenguaje (fuera del de máquina) disponibles para escribir programas en una computadora.

MENU: Conjunto de opciones que realizan una tarea.

MODIFICAR: Permite realizar cambios en los de los registros.

MODULO: Es un conjunto de procesos agrupados, con un objetivo en común.

MS-DOS: Medio ambiente con el cual trabaja la computadora.

OBJETO: Es una entidad que tiene una función específica de la cual es responsable y la información necesaria para cumplir con esa tarea. Es aquel que tiene un estado (características que lo hacen único), comportamiento (las operaciones que pueden ser invocadas por el objeto, como sus métodos en SMALLTALT) e identidad (es lo que lo distingue de todos los demás objetos).

PARADIGMA: (Latín paradigma, Griego paradigma) es una palabra que significó en su forma original, ejemplo ilustrativo. Conjunto de teorías, estándares y métodos que en un conjunto representan una manera de concebir el conocimiento.

PASSWORD: Es una clave de acceso que se determina con el fin de que no cualquier persona pueda tener acceso a determinado proceso.

POLIMORFISMO: Capacidad que tiene una clase de responder de manera diferente a un mismo mensaje. El polimorfismo permite crear clases que respondan a un solo mensaje.

PROGRAMA: Programar significa prefijar y describir un conjunto de criterios y acciones que deben regular el comportamiento de un sistema.

PROGRAMACION ESTRUCTURADA: Forma de programar que se basa en limitar el conjunto de estructuras posibles de un ordinograma a unas pocas estructuras privilegiadas, que, en general, tiene a estructurar el problema, procurando que su texto corresponda a su orden de ejecución.

PROGRAMACION MODULAR: Es un método de diseño y tiende a dividir el problema total en partes perfectamente diferenciadas que pueden ser analizadas, programadas y puestas a punto por separado. Se basa en el diseño descendente (Top_down).

PROGRAMACION ORIENTADA A OBJETOS: Es un método de diseño que se refiere a la técnica de construcción de sistemas, basada en la combinación de componentes, con límites bien establecidos, que enfatizan la creación de bibliotecas de componentes reutilizables.

REGISTRO: Es la forma organizada de agrupar varios campos.

SISTEMA: Cualquier conjunto de componentes cuya función conjunta es alcanzar un objetivo; un subsistema es un sistema que sirve como componente de un sistema mayor.

SOFTWARE: Las instrucciones que se utilicen para dirigir los componentes de hardware de la computadora para la realización de tareas determinadas; programas de computadora.

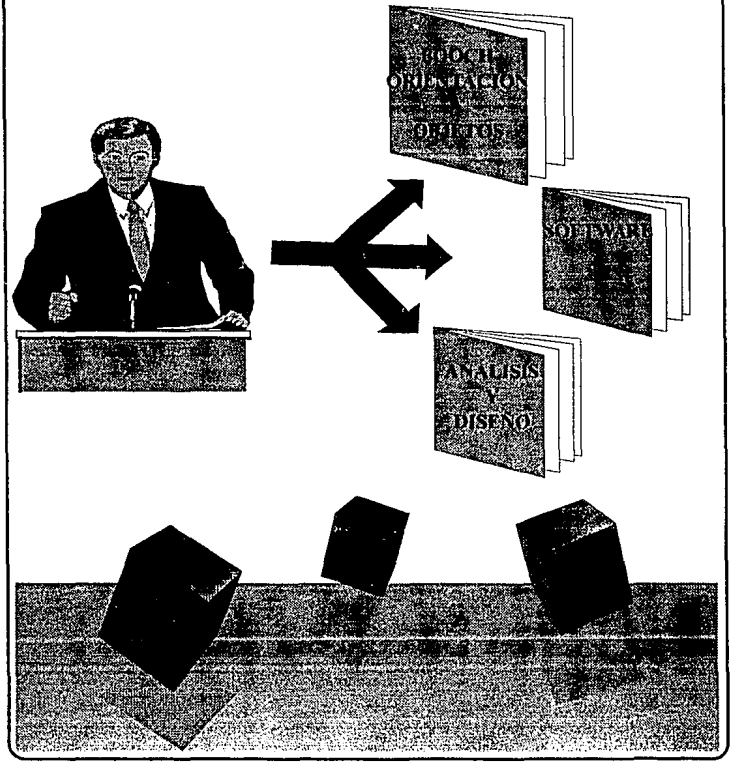
SOFTWARE DE APLICACIONES: Programas escritos para resolver problemas determinados de procesamiento de datos de una organización.

SOFTWARE DE SISTEMAS: Todo el software que no sea de aplicaciones; programas de lenguajes y sistemas operativos.

SUBDIRECTORIO: Es la división de un directorio cuya finalidad es la de organizar y clasificar información.

INDICE: Es una estructura organizada para accesar rápidamente a un registro dentro de un archivo.

BIBLIOGRAFIA



BIBLIOGRAFIA

Wiederhold Gio.
Diseño de Base de Datos.
Edit. MacGraw-Hill.
2da. Edición.
Año de impresión 1991.

Oktaba Hanna.
Diseño Orientado a Objetos, Método de Booch.
Notas de la Maestría en Ciencias de la Computación.
Año de impresión 1993.

Werner, P.
Concepts and Paradigms of Object-Oriented Programing.
Año de impresión 1990.

Pressman Roger S.
Ingeniería del Software (Un enfoque Práctico).
Edit. McGraw-Hill.
2da. Edición.
Año de impresión 1990.

Fairley Richard E.
Ingeniería de Software.
Edit. McGraw-Hill.
2da. Edición.
Año de impresión 1992.

Aguado Juaquín.
Introducción a C++.
Notas de la Maestría en Ciencias de la Computación.
Año de impresión 1993.

Mora Luis José.
Introducción a la informática.
Edit. Trillas.
4ta. Edición.
Año de impresión 1992.

Setrag Khoshafian, Razmik Abnous.
Object Orientation, Concepts, Languages, Databases, User Interfaces.
Edit. Wiley
Año de impresión 1993.

Booch Grady.
Object-Oriented Design With Applications.
The Benjamin/Cumming Pub.Comp.
Año de impresión 1991.

Shlaer, S.
Object Lifecycles: Modeling the World in States.
Edit. Prentice Hall
Año de impresión 1992.

Taylor David A.
Object Oriented Technology a Manager Guide.
Ed Addison Wesley.
Año de impresión 1990.

Meyer, B.
Object-Oriented Software Construction.
Edit. Prentice Hall.
Año de impresión 1988.

Padwick Gordon.
Paradox for Windows.
Edit. Ventura.
1era. Edición.
Año de impresión 1993.

Paradox for Windows, Guide to ObjectPAL.
Borland International.
Año de impresión 1993.

HEMEROGRAFIA

Journal of Object-Oriented Programming.
Revista de computación.
Mayo 1993.

Soluciones Avanzadas.
Revista de Tecnologías de Información y Estrategias de Negocios.
Volúmenes: Abril-Mayo 1993, Septiembre-Octubre, Noviembre-Diciembre 1993.