EVALUACION DEL PERSONAL DOCENTE

CURSO: HERRAMIENTA CASE
FECHA: 23 MAR-06 ABR

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACION CON EL ASISTENTE	PUNTUALIDAD
ING. LAURA SANDOVAL				

EVALUACION DE LA ENSEÑANZA

ORGANIZACION Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD LOGRADO EN EL CURSO	
ACTUALIZACION DEL CURSO	
APLICACION PRACTICA DEL CURSO	

EVALUACION DEL CURSO

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

ESCALA DE EVALUACION:- 1 A 10

1.- ¿LE AGRADO SU ESTANCIA EN LA DIVISION DE EDUCACION CONTINUA?

SI	NO

SI INDICA QUE "NO" DIGA PORQUE.

COORDINACION CURSOS DE COMPUTO
CENTRO DE INFORMACIÓN Y DOCUMENTACION

2.- MEDIO A TRAVES DEL CUAL SE ENTERO DEL CURSO:

PERIODICO EXCELSIOR		FOLLETO ANUAL		GACETA UNAM		OTRO MEDIO	
PERIODICO EL UNIVERSAL		FOLLETO DEL CURSO		REVISTAS TECNICAS			

3.- ¿QUE CAMBIOS SUGERIRIA AL CURSO PARA MEJORARLO?

_____

_____

4.- ¿RECOMENDARIA EL CURSO A OTRA(S) PERSONA(S)?

SI		**NO**	

5.- ¿QUE CURSOS LE SERVIRIA QUE PROGRAMARA LA DIVISION DE EDUCACION CONTINUA.?

_____

_____

6.- OTRAS SUGERENCIAS:

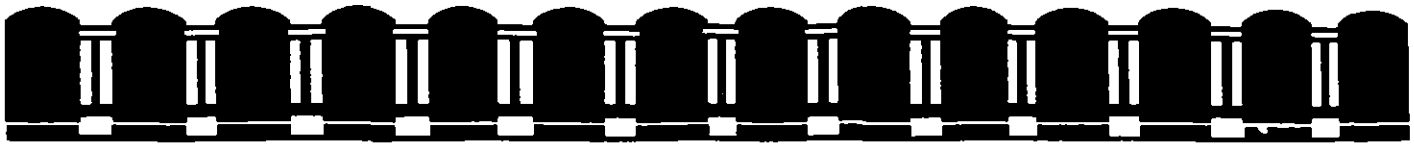
_____

_____

7.- ¿EN QUE HORARIO LE SERIA CONVENIENTE SE IMPARTIERAN LOS CURSOS DE LA DIVISION DE EDUCACION CONTINUA?
MARQUE EL HORARIO DE SU AGRADO

LUNES A VIERNES DE 16 A 20 HORAS	**MARTES Y JUEVES DE 17 A 21 HS SABADO DE 10 A 14 HS.**	**OTRO**
LUNES, MIERCOLES Y VIERNES DE 17 A 21 HORAS	**VIERNES DE 17 A 21 HS. SABADOS DE 10 A 14 HS**	

COORDINACION CURSOS DE COMPUTO
CENTRO DE INFORMACIÓN Y DOCUMENTACION



FACULTAD DE INGENIERIA UNAM
DIVISION DE EDUCACION CONTINUA

MATERIAL DIDACTICO

HERRAMIENTA CASE EN EL

DESARROLLO DE SISTEMAS

MARZO-ABRIL 1995

Palacio de Minería Calle de Tacuba 5 1er. Piso Delegación Cuauhtémoc 06000 México D.F. Tel. 521-40-20 Apdo. Postal M-2285

ANALISIS Y DISEÑO DE SISTEMAS DE INFORMACION

NOTAS ELABORADAS POR:

ING. ABANIA URIOSTEGUI MONDRAGON
ING. LAURA SANDOVAL MONTAÑO
ING. MA. TERESA SORIANO RAMIREZ

ANALISIS Y DISEÑO DE SISTEMAS DE COMPUTO

TEMA 1. ANALISIS DEL SISTEMA

1.1 Antecedentes

Es común, al hablar de la comunicación, señalar el vertiginoso e impresionante progreso logrado en las pocas décadas de su existencia. El contexto en el que se ha desarrollado el software está fuertemente ligado a las cuatro décadas de evolución de los sistemas informáticos. Un mejor rendimiento del hardware, un tamaño más pequeño y un costo más bajo, han dado lugar a sistemas informáticos más sofisticados.

La figura 1 describe la evolución del software dentro del contexto de las áreas de aplicación de los sistemas basados en computadoras.Durante los primeros años el hardware (fig 1a) era de propósito general, por otra parte el software se diseñaba a medida para cada aplicación y tenía una distribución relativamente pequeña.

La mayoría del software se desarrollaba y utilizaba por la misma persona u organización, debido a este entorno personalizado del software, el diseño era un proceso implícito, ejecutado en la cabeza de alguien y, la documentación no existía normalmente.

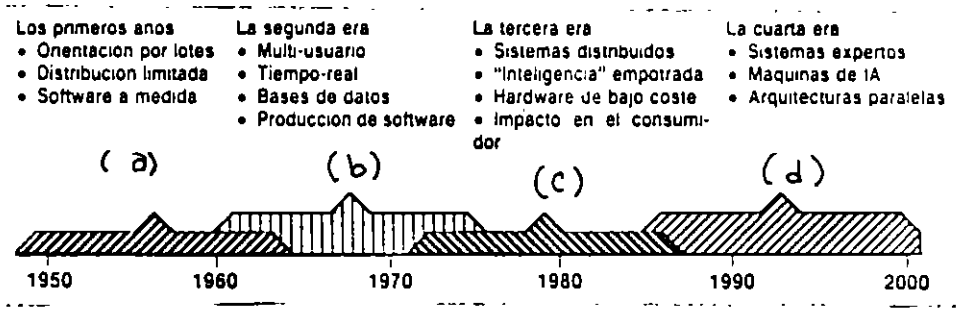


Fig 1.

La segunda era de la evolución de los sistemas de computadoras (fig 1b) se extiende desde la mitad de la década de los 60 hasta finales de los 70. La multiprogramación, los sistemas multiusuarios, introdujeron nuevos conceptos de interacción hombre-máquina.Las técnicas interactivas abrieron un nuevo mundo de aplicaciones, los sistemas podían reconocer, analizar y transformar datos de múltiples fuentes, controlando así los proceso y produciendo salidas en milisegundos en vez de minutos.

La tercera era (fig 1c) de la evolución de los sistemas de computadoras comenzó a mediados de los 70. El sistema distribuido (en este caso computadoras múltiples, cada una ejecutando funciones concurrentes y comunicándose con alguna otra) incremento notablemente la complejidad de los sistemas informáticos.

Redes de área local y global, comunicaciones digitales de alto ancho de banda,la llegada y amplio uso de los microprocesadores y computadoras personales, supusieron una fuerte presión sobre los desarrolladores de software.

La cuarta era (fig 1d) en software de computadoras ha comenzando. Las técnicas de la cuarta generación para el desarrollo de software están cambiando la forma en que algunos segmentos de la comunidad informática construye los programas de computadora. Los sistemas expertos y el software de inteligencia artificial se han trasladado finalmente del laboratorio a aplicaciones prácticas, en un amplio rango de problemas del mundo real.

Es así como el software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos.

1.2 DEFINICION DE SISTEMAS Y CARACTERISTICAS.

La palabra sistema es posiblemente el término más sobreutilizado y del que más se ha abusado en el léxico técnico. Hablamos de sistemas políticos y educativos, sistemas bancarios y de ferrocarril. La palabra no dice poco.

Podemos definir un sistema como un conjunto u ordenación de elementos organizados para llevar a cabo algún método, procedimiento o control mediante el procesamiento de información.

Dependiendo a la forma en la que se analizan los sistemas se tienen dos tipos de sistemas :

- **Abierto** : En este caso se deja al libre albedrío de la persona encargada.

- **Cerrado** : Se delimitan fronteras a partir de las cuales se trabajará, de tal forma que aquello que se encuentre fuera de las fronteras no afectará al sistema.

Conforme al comportamiento que presentan los sistemas se tiene la clasificación siguiente :

- **Determinístico** : El comportamiento que presenta el sistema es siempre el mismo.

- **Equifinalidad** : En este caso sea cual sea el estado inicial del sistema siempre se llegará a un estado final único.

- **Homeostático** : Para este caso los sistemas que se tienen siempre se están retroalimentando.

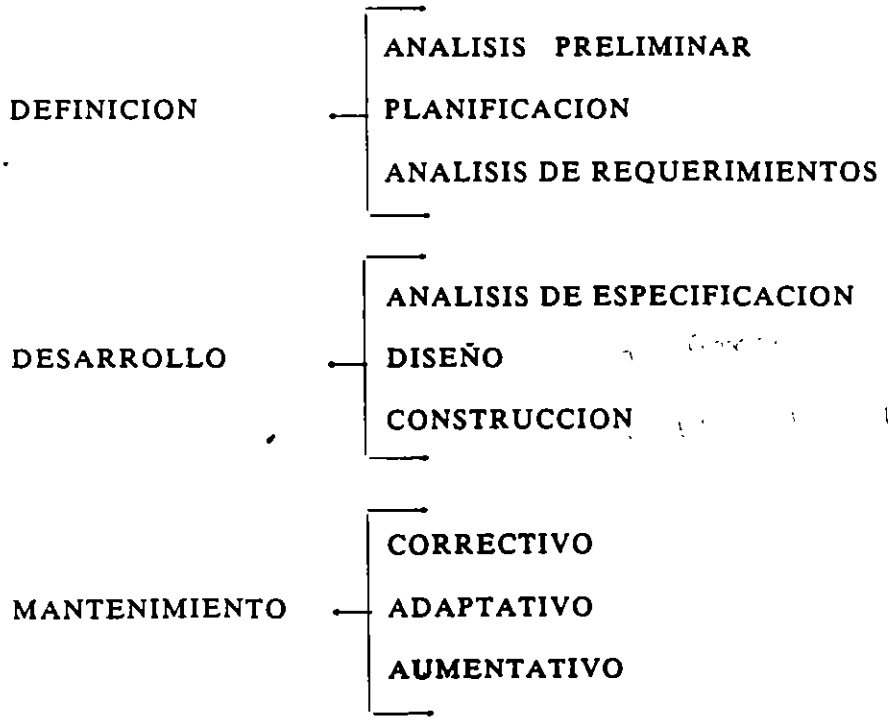
- **Teológico** : Este comportamiento se observa en todos los sistemas porque mantiene un proceso y persigue un objetivo.

1.3 CICLOS DE VIDA

Los procedimientos de la ingenieria del software son la cola que pega a los métodos y herramientas y facilita un desarrollo racional y oportuno del software de computadora. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, formas,etc.) que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios y las guías que facilitan a los gestores del software establecer su desarrollo.

El ciclo de vida es aquel que requiere un enfoque sistemático, secuencial, del desarrollo del software que comienza en el nivel del sistema y progresa a través de sus diversas fases.

Cada ciclo de vida tiene las fases siguientes :



ANALISIS Y DISEÑO DE SISTEMAS DE COMPUTO
TEMA I. ANALISIS DEL SISTEMA
LECTURA: FASES DEL CICLO DE VIDA.

El proceso de desarrollo del software contiene tres fases genéricas independientemente del paradigma de ingeniería elegido. Las tres fases, definición, desarrollo y mantenimiento, se encuentran en todos los desarrollos de software, independientemente del área de aplicación, tamaño del proyecto o complejidad.

La fase de definición se enfoca sobre el qué. Esto es, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué ligaduras de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto. Por tanto, han de identificarse los requerimientos claves del sistema y del software. Aunque los métodos aplicados durante la fase de definición variarán dependiendo del paradigma de ingeniería del software aplicado, de alguna forma se producirán tres pasos específicos:

Análisis del sistema. El análisis del sistema define el papel de cada elemento de un sistema informático, asignando finalmente el papel que jugará el software.

Planificación del proyecto. Una vez que está asignado el ámbito del software, se asignan los recursos, se estiman los costes y se definen las tareas y planificación del trabajo.

Análisis de requerimientos. El ámbito definido para el software de la dirección, pero antes de comenzar a trabajar, es necesario disponer de una información detallada del dominio de la información y de la función del software.

La fase de desarrollo se enfoca sobre el cómo. Esto es, durante el desarrollo, el que desarrolla el software intenta descubrir cómo han de diseñarse las estructuras de datos y arquitectura del software, cómo han de implementarse los detalles procedimentales, cómo ha de trasladarse el diseño a un lenguaje de programación y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo variarán dependiendo del paradigma de ingeniería del software aplicado. Sin embargo, de alguna forma se producirán tres pasos concretos:

Diseño del software. El diseño traslada los requerimientos del software a un conjunto de representaciones (algunas gráficas, otras tabulares o basadas en lenguajes) que describen la estructura de datos, arquitectura y procedimiento algorítmico.

Codificación. Las representaciones del diseño deben trasladarse a un lenguaje artificial, que da como resultados unas instrucciones ejecutables por la computadora. El paso de la codificación ejecuta esta traslación.

Prueba del software. Una vez que el software se ha implementado en una forma ejecutable por la máquina, debe ser probado para descubrir los defectos que puedan existir en la función, lógica e implementación.

La fase de mantenimiento se enfoca sobre el cambio que va asociado con una corrección de errores, adaptaciones requeridas por la evolución del entorno del software y modificaciones debidas a los cambios de los requerimientos del cliente para reforzar o aumentar el sistema. La fase de mantenimiento replica los pasos de las fases

de definición y desarrollo, pero en el contexto del software existente. Durante la fase de mantenimiento se encuentran tres tipos de cambios:

Corrección. Incluso con las mejores actividades para garantizar la calidad, es probable que el cliente descubra defectos en el software. El mantenimiento correctivo cambia el software para corregir los defectos.

Adaptación. Con el paso del tiempo es probable que cambie el entorno original (por ejemplo, CPU, sistema operativo, periféricos) para el cual se desarrolló el software. El mantenimiento adaptativo se traduce en modificación del software para acomodarlo a los cambios de su entorno externo.

Aumento. Conforme se utiliza el software, el cliente/usuario reconocerá funciones adicionales que podría ser beneficioso añadirlas.

Las fases y pasos relacionados descritos en esta visión genérica de la ingeniería del software, se complementan con varias actividades protectoras. Las revisiones se realizan durante cada paso para asegurar que se mantiene la calidad. La documentación se desarrolla y controla para asegurar que toda la información sobre el sistema y software estarán disponibles para un uso posterior. El control de los cambios se instituye de forma que los cambios puedan ser mejorados y registrados.

El método en cada caso puede variar de un paradigma a otro, pero el enfoque global que exige la definición, desarrollo y mantenimiento permanece invariable. Se puede realizar con disciplina y métodos bien definidos o de forma completamente desordenada. Pero habrá que realizarlos de alguna forma.

TEMA 2. ANALISIS DE REQUERIMIENTOS

Subtema 2.1 Fundamentos del Análisis de requerimientos

Para realizar bien el desarrollo de software es esencial realizar una especificación completa de los requerimientos del mismo.

Es importante realizar una especificación completa de los requerimientos del software, la tarea de análisis de requerimientos es un proceso de descubrimiento y refinamiento, de ahí la importancia de una buena especificación.

El papel del desarrollador de software y del cliente es un papel activo, el cliente intenta reformular su concepto de la función y comportamiento de los programas en detalles concretos, el que desarrolla el software actúa como interrogador, consultor y el que resuelve los problemas.

El analista.

El analista ejecuta o coordina cada una de las tareas asociadas con el análisis de los requerimientos de software.

El analista debe exhibir los siguientes rasgos de carácter :

- Habilidad para comprender conceptos abstractos, reorganizarlos en divisiones lógicas y sintetizar "soluciones" basadas en cada división.

- Habilidad para entresacar hechos importantes de fuentes conflictivas o confusas.

- Habilidad para comprender entornos de usuario/cliente.

- Habilidad para aplicar elementos hardware y/o software a entornos de usuario/cliente.

- Habilidad par comunicarse bien en forma escrita y verbal.

Los individuos que se paran excesivamente en los detalles con frecuencia pierden de vista el objetivo global de los programas. Los requerimientos del software deben ser descubiertos de una manera "descendente"; las funciones importantes, interfaces e información deben comprenderse completamente antes de que se especifiquen los detalles de las capas sucesivas.

Los problemas subyacentes al análisis de requerimientos son atribuibles a muchas causas:

- Pobre comunicación : Esto hace difícil la adquisición de la información.

- Técnicas y herramientas inadecuadas : Esto produce especificaciones inadecuadas o imprecisas.

- Tendencia a acortar el análisis de requerimientos : Conduce a un análisis inestable.

- Un fallo en considerar alternativas antes de que se especifique el software.

Principios del 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 subdividirse en forma que se descubran los detalles de una manera progresiva (o jerárquica).

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

Haciendo referencia un poco al dominio de la información. Todas las aplicaciones del software pueden colectivamente llamarse procesamiento de datos. Este término contiene la clave de lo que entendemos por requerimientos del software. El software se construye para procesar datos, pero transformar datos de una forma a otra; esto es, para aceptar entrada, manipularla de alguna forma y producir una salida.

Vayamos ahora a la construcción de prototipos del software

En algunos casos es posible aplicar los principios de análisis fundamental y derivar a una especificación en papel del software desde el cual pueda desarrollarse un diseño. En otras situaciones, se va a una recolección de los requerimientos, se aplican los principios de análisis y se construye un modelo de software, llamado prototipo, según las apreciaciones del cliente y del que lo desarrolla. Finalmente, hay circunstancias que requieren la construcción de un prototipo al comienzo del análisis, puesto que el modelo es el único medio mediante el cual los requerimientos pueden ser derivados efectivamente.

Perfil de las especificaciones de los Requerimientos del Software.

La especificación de requerimientos de software se produce en la culminación de la tarea de análisis. La función y comportamiento asignados al software como parte de la ingenieria de sistemas se refina estableciendo una descripción completa de la información, una descripción funcional detalla, una indicación de los requerimientos de rendimiento y las ligaduras de diseño, unos criterios de validación apropiados y otros datos pertinentes a los requerimientos.

Subtema 2.2 Métodos de Análisis de Requerimientos.

Las metodologías de análisis de requerimientos combinan procedimientos sistemáticos con una notación única para analizar los dominios de información y funcionalidad de un problema de software. En esencia, los métodos de análisis de requerimientos del software, facilitan al ingeniero de software aplicar principios de análisis fundamentales, dentro del contexto de un método bien definido.

Todos los métodos pueden ser evaluados en el contexto de las siguientes características comunes :

1) mecanismos para el análisis del dominio de la información

2) método de representación funcional

3) definición de interfaces

4) mecanismos para subdividir el problema

5) soporte de la abstracción

6) representación de las visiones físicas y lógicas

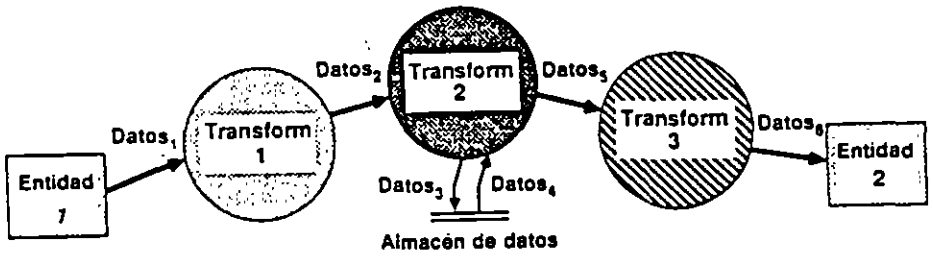
La mayoría de los métodos de análisis permiten al analista evaluar la representación física de un problema antes de derivar a la solución lógica.

Métodos de análisis orientados al flujo de datos.

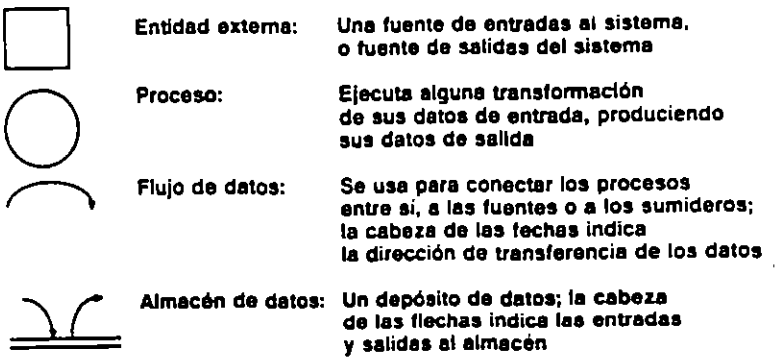
Flujo de Información.

(fig 5.1) Una técnica para representar el flujo de información a través del sistema basado en computadora lo podemos ver en esta figura. La función global del sistema se representa como una transformación sencilla de la información, representada en la figura como una burbuja. Una o más entradas, representadas como flechas con etiquetas, conducen la transformación para producir la información de salida.

Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida.



(fig 5.2) La forma básica de un DFD lo podemos ver en esta figura. El DFD puede usarse para representar un sistema o software a cualquier nivel de abstracción. Además pueden particionarse en niveles que representan flujo incremental de información y detalle funcional.



(fig 5.3) La simbología de un DFD la podemos ver en esta figura. Se utiliza un rectángulo pará representar una entidad externa, esto es, un elemento del sistema (por ejemplo una persona). Un círculo representa un proceso o transformación que se aplica a los datos y que los cambia de alguna forma. Una flecha representa uno o más elementos de datos. Todas las flechas de un diagrama de flujo de datos deben estar etiquetadas. Una doble línea representa un almacenamiento de datos (información almacenada que es usada por el programa). Debido a la excepcional simplicidad de la simbología del DFD es una de las razones por las que las técnicas de análisis orientadas al flujo de datos son tan ampliamente usadas.

Métodos orientados a la estructura de datos.

Puesto que el dominio de la información para un problema de software comprende el flujo de datos, el contenido de datos y la estructura de datos. Los métodos de análisis orientados a la estructura de datos representan los requerimientos del software enfocándose hacia la estructura de datos en vez de al flujos de datos. Aunque cada método orientado a la estructura de datos tiene un enfoque y notación distinta, todos tienen algunas características en común como son:

Primero, todos asisten al analista en la identificación de los objetos de información clave (también llamados entidades) y operaciones (también llamadas acciones o procesos)

Segundo, todos suponen que la estructura de la información es jerárquica

Tercero, todos requieren que la estructura de datos se represente usando la secuencia, selección y repetición. (vistas anteriormente)

y cuarto, todos dan un conjunto de pasos para transformar una estructura de datos jerárquica en una estructura de programa.

Dentro de los métodos orientados a la estructura de datos, el desarrollo de sistemas estructurados de datos (DSED), también llamado metodología de Warnier-Orr, se basa en el trabajo sobre análisis del dominio de información, realizado por J. D. Warnier. Warnier desarrollo una notación para representar la jerarquía de la información usando las tres construcciones :

1) Secuencia

2) Selección

3) Repetición

El diagrama de Warnier facilita al analista representar jerarquías de información de una manera compacta. Se analiza el dominio de la información y representa la naturaleza jerárquica de la salida.

El método, comprende todos los atributos del dominio de información: flujo, contenido y estructura de datos. Para ilustrar la notación y dar una visión general del método de análisis.

Requerimientos de las bases de datos

El término de base de datos actualmente se ha convertido en uno de los muchos tópicos del campo de las computadoras. Podemos definir una base de datos como una colección de información organizada de forma que facilita el acceso, análisis y creación de informes.

El análisis de requerimientos para una base de datos incorpora las mismas tareas que el análisis de requerimientos del software. Es necesario un contacto estrecho con el cliente; es esencial la identificación de las funciones e interfaces; se requiere la especificación del flujo, estructura y asociatividad de la información y debe desarrollarse un documento formal de los requerimientos.

TEMA II. ANALISIS DE REQUERIMIENTOS
LECTURA: AREAS DEL ANALISIS DE REQUERIMIENTOS.
El análisis de requerimientos puede dividirse en cuatro áreas:

1) reconocimiento del problema.
2) evaluación y síntesis
3) especificación y
4) revisión

Inicialmente, el analista estudia la especificación del sistema (si existe) y el Plan del Proyecto. Es importante comprender el contexto del sistema y revisar el ámbito de los programas que se uso para generar las estimaciones de la planificación. Posteriormente, debe establecerse la comunicación necesaria para el análisis, de forma que se asegure el reconocimiento del problema.
El analista debe establecer contacto con el equipo técnico y de gestión del usuario/cliente y con la empresa que vaya a desarrollar el software. El gestor del programa puede servir como coordinador para facilitar el establecimiento de los caminos de comunicación.
El objetivo del analista es reconocer los elementos básicos del programa tal como lo percibe el usuario/cliente.
La evaluación del problema y la síntesis de la solución es la siguiente área principal de trabajo del análisis. El analista debe evaluar el flujo y estructura de la información, refinar en detalle todas las funciones del programa en detalle, establecer las características de la interface del sistema y descubrir las ligaduras del diseño. Cada una de las tareas sirven para describir el problema de forma que pueda sintetizarse un enfoque o solución global.
Por ejemplo, un importante suministrador de materiales de fontanería necesita un sistema de control de inventario. El analista encuentra que los problemas con el sistema manual actual incluyen:
1) imposibilidad de obtener rápidamente el estado de una componente.
2) dos o tres días de tiempo medio para actualizar un archivo de tarjetas y
3) múltiples reórdenes del mismo vendedor debido a que no hay forma de asociar vendedores como componentes, etc. Una vez que se han identificado los problemas, el analista determina qué información va a ser producida por el nuevo sistema y qué datos se le suministrarán al sistema. Por ejemplo, el cliente desea un informe diario que indique los elementos que se han tomado del inventario y cuántos elementos similares permanecen. el cliente indica que los empleados del invéntario registrarán el número de identificación de cada pieza cuando dejen el área de inventario.
Hasta la evaluación de los problemas actuales y de la información deseada (entrada y salida), el analista comienza por sintetizar una o más soluciones. Un sistema basado en un terminal en línea resolverá varios problemas, pero parece que se va a necesitar un sistema de gestión de base de datos, pero ¿está justificada la necesidad de asociatividad el usuario/cliente? El proceso de evaluación y síntesis continúa hasta que el analista y el cliente creen que el software puede ser especificado adecuadamente para la fase de desarrollo.
En una etapa no es posible hacer una especificación detallada. El

cliente puede no estar seguro de precisar bien lo que quiere. El que desarrolla el software puede no estar seguro de que un enfoque concreto sea apropiado para realizar la función y comportamiento deseado. Por éstas razones puede presentarse un enfoque alternativo, llamado construcción de prototipos, para el análisis de requerimientos.(Se tratará la construcción de prototipos más adelante).

Las tareas asociadas con el análisis y especificación existen para dar una representación del programa que pueda ser revisada y aprobada por el cliente. En un mundo ideal el cliente desarrolla una Especificación de Requerimientos del Software completamente por sí mismo. Esto se presenta raramente en el mundo real. En el caso mejor, la especificación se desarrolla conjuntamente entre el cliente y el técnico.

Una vez que se hayan descrito las funciones básicas, comportamiento, interface e información, se especifican los criterios de validación para demostrar una comprensión de una correcta implementación de los programas. Estos criterios sirven como base para hacer la prueba durante el desarrollo de los programas. Para definir las características y atributos del software se escribe una especificación de requerimientos formal. Además, para los casos en los que se desarrolle un prototipo se realiza un Manual de Usuario Preliminar.

Puede parecer innecesario desarrollar un manual de usuario en una etapa tan temprana del proceso de desarrollo. Pero de hecho, este borrador de manual de usuario fuerza al analista a tomar el punto de vista del usuario del software. El manual permite al usuario/cliente revisar el software desde una perspectiva de ingeniería humana y frecuentemente produce el comentario: "La idea es correcta, pero ésta no es la forma en que pensé que se podría hacer esto". Es mejor descubrir tales comentarios lo más tempranamente posible en el proceso.

Los documentos del análisis de requerimientos (especificación y manual de usuario) sirven como base para una revisión conducida por el cliente y el técnico. La revisión de los requerimientos casi siempre produce modificaciones en la función, comportamiento, representación de la información, ligaduras o criterios de validación. Además, se realiza una nueva apreciación del Plan del Proyecto Software para determinar si las primeras estimaciones siguen siendo válidas después del conocimiento adicional obtenido durante el análisis.

LECTURA: ANALISIS ORIENTADO AL OBJETO.

El diseño orientado al objeto (DOO), 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 DOO 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 al objeto está ligada a su habilidad para construir, basándose entres 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 DOO da un mecanismo que facilita al diseñador adquirir los tres sin complejidad o compromiso.

Orígenes del diseño orientado al objeto.

Los objetivos y las operaciones no son un nuevo concepto de programación, pero sí lo es el diseño orientado al objeto. En los primeros días de la computación, los lenguajes ensambladores facilitaban a los programadores la utilización de las instrucciones máquina (operadores) para manipular los elementos de datos (operandos). El nivel de abstracción que se aplicaba al dominio de la solución era muy bajo.

Conforme aparecieron los lenguajes de programación de alto nivel (por ejemplo Fortran, algol, cobol, etc) los objetos y operaciones del espacio de problemas del mundo real podían ser modelados mediante datos y estructuras de control predefinidas, que estaban disponibles como partes del lenguaje de alto nivel. En general, el diseño de software se enfocaba sobre la representación del detalle procedimental usando el lenguaje de programación elegido. Los conceptos de diseño, tales como refinamientos sucesivos de una función, modularidad procedimental y, posteriormente, programación estructurada, fueron introducidos entonces.

Durante los años 1970, se intrudujeron conceptos tales como abstracción y ocultación de la información y, emergieron métodos de diseño conducidos por los datos, pero los que desarrollaban software aún se encontraban sobre el proceso y su representación. Al mismo tiempo, los lenguajes de alto nivel modernos (por ejemplo Pascal) introdujeron una variedad mucho más rica de tipos y estructuras de datos.

Aunque los lenguajes de alto nivel convencionales (lenguajes a partir de Fortran y algol) evolucionaron durante los años 1960 y 1970, los investigadores estaban trabajando mucho sobre una nueva clase de lenguaje de simulación de prototipos, tales como Simula y Smalltalk. En estos lenguajes, la abstracción de datos tenía una gran importacia y los problemas del mundo real se representaban mediante un conjunto de objetos de datos, a los cuales se les añadía el correspondiente conjunto de operaciones. El uso de estos lenguajes era radicalmente diferente del uso de los lenguajes más convencionales.

El método del diseño orientado al objeto ha emergido durante los últimos 17 años. Los primeros trabajos en diseño de software pusieron la base estableciendo la importancia de la abstracción,

ocultación de la información y de la modularidad en la calidad del software. En algunos aspectos, los métodos de diseño orientados a estructuras de datos tales como Desarrollo de Sistemas Estructurados de Datos (DSED) y Desarrollo de Sistema de Jackson (DSJ) , pueden verse como orientados al objeto.

Durante los años 1980 la rápida evolución de los lenguajes de programación Smalltalk y Ada causaron un creciente interés en DOO. En las primeras discusiones sobre el método para conseguir un diseño orientado al objeto. Abbott mostró "cómo el análisis de sentencias en lenguaje natural del problema y sus solución, pueden usarse como guía para desarrollar la parte visible de un paquete útil (un paquete útil que tenga los datos y procedimientos que operan sobre ellos) y el algoritmo particular para un problema dado". Booch extendió el trabajo de Abbott y ayudó a popularizar el concepto de diseño orientado al objeto. Hoy el DOO se está usando en aplicaciones de diseño de software que van desde animaciones de gráficos por computadora a las telecomunicaciones.

Conceptos del diseño orientado al objeto.

Como otros métodos de diseño, el DOO introduce un nuevo con junto de terminología, notación y procedimientos para la derivación de un diseño del software.

La función del software es relacionada cuando una estructura de datos (de distintos niveles de complejidad) es manipulada mediante uno o más procesos, de acuerdo con un procedimiento definido mediante un algoritmo estático y órdenes dinámicas. Para conseguir un diseño orientado al objeto, se debe establecer un mecanismo para:

1) la representación de la estructura de datos,
2) La especificación del proceso y
3) El procedimiento de llamada.

Un objeto es una componente del mundo real que se transforma en el dominio del 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, órdenes, archivos, visualizaciones, conmutadores, señales, cadenas alfanuméricas o cualquier otra persona, lugar o cosa. Cuando un objeto se transforma en una realización software, consta de una estructura de datos privada y procesos, llamados operaciones o métodos, que pueden transformar legítimamente la estructura de 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. La realización software de un objeto se muestra en la figura (fig 9.1).

Refiriéndose a la figura, un objeto del mundo real (un diccionario) se transforma en una realización software para un sistema basado en computadora. La realización software de un diccionario exhibe una estructura de datos privada y las operaciones relativas a ella. La estructura de datos puede tomar la forma descrita mediante diagramas de Warnier-Orr de la figura (fig 9.2). Las entradas del diccionario están compuestas de palabras, una guía de pronunciación y una o más definiciones. Un dibujo o diagrama puede estar también contenido dentro de una entrada. El diccionario de objetos también

contiene un conjunto de operaciones que puede procesar los elementos de la estructura de datos. La parte privada de un objeto es la estructura de datos y el conjunto de operaciones para la estructura de datos.
Un objeto tiene también una parte compartida que es su interfaz. Los mensajes se mueven a través de la interfaz y especifican qué operaciones del objeto se desean, pero no cómo se va a realizar la operación. El objeto que recibe un mensaje determina cómo se implementa la operación solicitada.
Definiendo un objeto con una parte privada y dando mensajes para llamar al procesamiento adecuado, se consigue la ocultación de la información; esto es, los detalles de la implementación están ocultos a todos los elementos del programa exteriores al objeto. Los objetivos y sus operaciones dan una modularidad inherente; esto es, los elementos del software (datos y procesos) se agrupan junto con los mecanismos de interfaces bien definidos.
De esta forma se tiene que el análisis orientado al objeto puede describirse de la siguiente forma:

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 objetivos se determinan subrayando cada nombre o cláusula nominal e introduciéndolo en una tabla sencilla. Deben anotarse los sinónimos. Si se requiere que el objeto se implemente 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 indica un test 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).

Para ilustrar el método de análisis orientado al objeto, considere al siguiente texto:

> El software SCCT recibirá la información de entrada de un lector de código de barras a intervalos de tiempo que estén de acuerdo con la velocidad de la cinta transportadora. Los datos del código de barras se decodificarán en un formato de identificación de caja. El software examinará una base de datos de unas 1000 entradas para determinar la posición del cubo apropiado para la caja que actualmente está en el lector (estación de clasificación). SE usará una lista FIFO para tomar nota de las posiciones de envío de cada caja conforme se mueve por la estación de clasificación.

El software SCCT recibirá también entrada de un tacómetro de pulsos que se usará para sincronizar la señal de control con el mecanismo de envío. Basándose en el número de pulsos que se generarán entre la estación de clasificación y el envío, el software producirá una señal de control a la estación de envío indicando la posición correspondiente de la caja...

Analizando un poco el texto anterior muchos de los nombres subrayados indican que residen en el espacio del problema y n o son necesarios para la implementación del programa. Otros nombres, tales como datos del código de barra, posición del cubo y señal de control, residen en el espacio de la solución e introducen en la tabla de objetos y operaciones. Siguiendo el procedimiento de análisis orientado al objeto, se subrayan a continuación todos lo adjetivos y se colocan en la tabla como se muestra en ella.
Cuando uno de los objetivos representa un elemento de información que debe ser procesado por el programa. Para determinar las funciones de procesamiento, debemos aislar todas las operaciones. Por lo que siguiendo el procedimiento de análisis:

El software SCCT recibirá la información de entrada de un lector de código de barras a intervalos de tiempo conforme a la velocidad de la cinta transportadora. Los datos del código de barras se decodificarán en un formato de identificación de caja. El software examinará una base de datos de unas 1000 entradas para determinar la posición apropiada del cubo para la caja que actualmente está en el lector (estación de clasificación). Se usará una lista FIFO para tomar nota de las posiciones de envío de cada caja conforme se mueve por la estación de clasificación.
El software SCCT recibirá también entrada de un tacómetro de pulsos que se usará para sincronizar la señal de control con el mecanismo de envío. Basándose en el número de pulsos que se generarán entre la estación de ordenación envío, el software producirá una señal de control a la estación de envío indicando la posición correspondiente de la caja...

TABLA
OBJETOS Y OPERACIONES

Objeto	Atributo	Operación correspondiente
Datos	Código de caja	Decodificador
Posición	Caja	Determinar
Señal	Control	Sincronizar, producir
Entrada	Tac. pulso	Recibir
Lista	FIFO	Tomar nota
Posiciones	Envío	Tomar nota
Intervalos	Tiempo	Conformar
Formato	Id. caja	Decodificar
Base de datos	1000 entradas	Examinar
Pulsos	Tac.	Generado

Algunas de las operaciones anotadas pertenecen al espacio de solución (por ejemplo recibir, buscar, producir) mientras que otras son parte del espacio del problema (por ejemplo sincroniza). Las operaciones del espacio de solución se añaden a la tabla de objetos y operaciones.

LECTURA: CONSTRUCCION DE PROTOTIPOS.

El análisis debe ser conducido independientemente del paradigma de ingeniería de software aplicado. Sin embargo, la forma que ese análisis tomará puede variar. En algunos casos es posible aplicar los principios de análisis fundamental y derivar a una especificación en papel del software desde el cual pueda desarrollarse un diseño. En otras situaciones, se va a una recolección de los requerimientos, se aplican los principios de análisis y se construye un modelo de software, llamado un prototipo, según las apreciaciones del cliente y del que lo desarrolla. Finalmente, hay circunstancias que requieren la construcción de un prototipo al comienzo del análisis, puesto que el modelo es el único medio mediante el que los requerimientos pueden ser derivados efectivamente.

El modelo entonces se transforma en una producción de software. En muchos casos (aunque no todos), la construcción de un prototipo, acoplado posiblemente con métodos de análisis sistemáticos, es una forma efectiva de ingeniería del software.

Todos los proyectos de ingeniería del software comienzan con una petición del cliente. La petición puede estar en la forma de una memoria que describe un problema, un informe que define un conjunto de objetivos comerciales o del producto, una Petición de Propuesta formal de una agencia o compañía exterior, o una Especificación del Sistema que ha asignado una función y comportamiento al software, como un elemento de un sistema basado en computadora. Suponiendo que existe una petición para un programa de una de las formas para construir un prototipo del software se aplican los siguientes pasos:

PASO 1. Evaluar la petición del software y determinar si el programa a desarrollar es un buen candidato para construir un prototipo. No todos los programas son adecuados para la construcción de prototipos. Varios factores de candidatura de construcción de prototipos pueden ser definidos: área de aplicación, complejidad, de la aplicación, características del cliente y características del proyecto.

Debido a que el cliente debe interaccionar con el prototipo en los últimos pasos, es esencial que:

1) El cliente participe en la evaluación y refinamiento del prototipo.
2) El cliente sea capaz de tomar decisiones de requerimientos de una forma oportuna.

Finalmente, la naturaleza del proyecto de desarrollo tendrá una fuerte influencia en la eficacia del prototipo.

PASO 2. Dado un proyecto candidato aceptable, el analista desarrolla una representación abreviada de los requerimientos. Antes de que pueda comenzar la construcción de un prototipo, el analista debe representar los dominios funcionales y de información del programa y desarrollar un método razonable de partición.

PASO 3. Después de que se haya revisado la representación de los requerimientos, se crea un conjunto de especificaciones de diseño abreviadas para el prototipo. El diseño debe ocurrir antes de que comience la construcción del prototipo. Sin embargo, el diseño de un prototipo se enfoca normalmente hacia la arquitectura a nivel superior y a los aspectos de diseño de datos, en vez de hacia el

diseño procedimental detallado.
PASO 4. El software del prototipo se crea, prueba y refina. Idealmente, los bloques de construcción de software preexistentes se utilizan para crear el prototipo de una forma rápida. Debido a que tales bloques construidos raramente existen. Alternativamente, pueden usarse herramientas de construcción de prototipos especializadas para asistir al analista/diseñador en la representación del diseño y traducirlo a una forma ejecutable.
Si la implementación de un prototipo que funcione es impracticable, el escenario de construcción de prototipos puede aún aplicarse. Para las aplicaciones interactivas con el hombre, es posible frecuentemente crear un prototipo en papel que describa la interacción hombre-máquina (preguntas, presentaciones, decisiones, etc.) usando una serie de hojas de historia. Cada hoja de historia contiene una representación de una imagen de la pantalla con texto, que describe la interacción entre la máquina y el usuario. El cliente revisa las horas de historia, obteniendo una perspectiva de usuario de la operación del software.
PASO 5. Una vez que el prototipo ha sido probado, se presenta al cliente, el cual "conduce la prueba" de la aplicación y sugiere modificaciones. Este paso es el núcleo del método de construcción de prototipo. Es aquí donde el cliente puede examinar una representación implementada de los requerimientos del programa, sugerir modificaciones que harán al programa cumplir mejor las necesidades reales.
PASO 6. Los pasos 4 y 5 se repiten iterativamente hasta que todos los requerimientos estén formalizados o hasta que el prototipo haya evolucionado hacia un sistema de producción. El paradigma de construcción del prototipo puede ser conducido con uno o dos objetivos en mente:

1) el propósito del prototipo es establecer un conjunto de requerimientos formales que pueden luego ser traducidos en la producción de programas mediante el uso de métodos y técnicas de ingeniería de programación, o
2) el propósito de la construcción del prototipo es suministrar un continuo que pueda conducir al desarrollo evolutivo de la producción del software. Ambos métodos tienen sus méritos y ambos crean problemas.

Métodos y herramientas para la construcción de prototipos.
Para que la construcción de prototipos de software sea efectivo, un prototipo debe desarrollar rápidamente, de forma que el cliente pueda comprobar los resultados y recomendar cambios. Para conseguir una construcción rápida de prototipo, existen tres clases genéricas de métodos y herramientas: Técnicas de la cuarta generación, componentes de software reusables, especificación formal y entornos de construcción de prototipo.
TECNICAS DE LA CUARTA GENERACION. Las técnicas de la cuarta generación (4GT) comprenden un amplio repertorio de lenguajes de informes y preguntas de base de datos, generadores de programas y aplicaciones y otros lenguajes no procedimentales de muy alto nivel. Debido a que las 4GT facilitan al ingeniero de programación general, código ejecutable rápidamente, son ideales para la construcción rápida de prototipos. Desafortunadamente, el dominio de aplicación de las 4GT está actualmente limitado a sistemas de

información comerciales.

COMPONENTES DE SOFTWARE REUSABLES. Otro método para la construcción rápida de prototipos es ensamblar, en vez de construir, el prototipo usando un conjunto de componentes software existente. Una componente software puede ser una estructura de datos (o base de datos) o una componente arquitectónica software (por ejemplo un programa o una componente procedimental (es decir, un módulo). En cada caso, la componente software debe ser diseñada en forma que facilite el ser reusada sin conocer los detalles de su funcionamiento interno. Debe observarse que un producto software existente puede ser usado como un prototipo para un "nuevo, mejorado" producto competitivo. De alguna forma, esto es una forma de reusabilidad para la construcción de prototipos de software.

ESPECIFICACION FORMAL Y ENTORNOS PARA LA CONSTRUCCION DE PROTOTIPOS. Se han desarrollado varios lenguajes de especificación formal para reemplazar las técnicas de especificación en lenguaje natural. Hoy, los desarrolladores de estos lenguajes formales están en el proceso de desarrollar entornos interactivos que:

1) facilite al analista crear interactivamente una especificación basada en el lenguaje de un sistema o software;
2) llame a herramientas automáticas que traduzcan las especificaciones basadas en lenguajes en código ejecutable y
3) faciliten al cliente utilizar el código ejecutable del prototipo para refinar los requerimientos formales.

La forma de especificar tiene mucho que ver con la calidad de la solución. Los ingenieros de software que se han esforzado en trabajar con especificaciones incompletas, inconsistentes o mal establecidas han experimentado la frustación y confusión que invariantemente se produce. Las consecuencias se padecen en la calidad, oportunidad y completitud del software resultante.

Las técnicas de análisis pueden conducir a una especificacion en papel que contenga las descripciones gráficas y el lenguaje natural de los requerimientos del software. La construcción de prototipos conduce a una especificación ejecutable, esto es, el prototipo sirve como una representación de los requerimientos. Los lenguajes de especificación formal conducen a representaciones formales de los requerimientos que pueden ser verificados o analizados.

LECTURA: BASES DE DATOS.
Un sistema de base de datos.
Se puede definir un sistema de base de datos como un sistema de mantenimiento de registros, basado en computadoras, es decir, un sistema que de manera general registra y mantiene información de importancia para la organización donde el sistema opera.
Un sistema de base de datos tiene cuatro componentes principales: datos, hardware, software y usuarios.

Datos
Se refieren a los valores registrados dentro de la base de datos. Es importante distinguir entre "datos" e información" ya que no siempre significan lo mismo. Un dato, simplemente es un valor (número, letras,...) mientras que la información es el significado de esos valores según el sentido que les dé el usuario. Los datos se dividen en una o más bases de datos, pero didácticamente es válido suponer que existe una sola base de datos.

Hardware
Se compone de los volúmenes de almacenamiento secundario (discos, tambores),...) donde se encuentra la base de datos junto con dispositivos asociados (unidades de control, canales,...).

Software
Es un sistema de administración de bases de datos o DBMS y existe entre el usuario y la base de datos física. Este DBMS maneja todas las solicitudes de acceso a la base de datos y protege a los usuarios contra los detalles a nivel hardware apoyando las operaciones del usuario.

Usuarios
Se pueden considerar diversos tipos de usuarios.
Existe el usuario programador de aplicaciones quien escribe programas de aplicación que utilicen bases de datos. Dichos programas recuperan información, crean información, suprimen o cambian información, y en general operan con los datos de todas las maneras posibles.
El usuario final es quien accesa la base de datos desde una terminal. Dicho acceso lo realiza a través de un lenguaje de consulta con el cual puede realizar todas las funciones de recuperación, creación, modificación y supresión de la información contenida en la base de datos.
El usuario administrador de Bases de Datos o DBA es quien organiza el sistema para un desempeño adecuado y le hace los ajustes necesarios al cambiar los requerimientos.

Objetivos de los sistemas de bases de datos
El objetivo primordial de un sistema de base de datos es proporcionar a la empresa un control centralizado de sus datos. Dicho control trae consigo diversas ventajas:

Control de la Redundancia.
Se evita que cada aplicación tenga sus propios archivos

(Redundancia=Información repetida). La Redundancia propicia un desperdicio del espacio de almacenamiento. En ocasiones, la redundancia no se elimina totalmente (por rabones técnicas o comerciales) sin embargo se controla y el sistema debe responsabilizarse de propagar las actualizaciones que se le hagan a los datos.

Evitar Inconsistencias.
Cuando en un sistema de bases de datos, existe redundancia, hay duplicidad de datos. Esta duplicidad puede originar que en determinado momento una actualización no se propague correctamente y se dé la inconsistencia de datos. En este momento, la información no será totalmente confiable, ya que, datos repetidos no son concordantes y la base de datos es inconsistente. Con lo anterior, se puede decir que la inconsistencia se puede evitar si se controla la redundancia.

Datos Compartidos.
Se pueden tener aplicaciones que compartan los datos de la base de datos, e incluso, aplicaciones que operen con los mismos datos almacenados. Con ellos, nuevas aplicaciones pueden realizarse sin tener que crear nuevos almacenamientos.

Seguridad.
Al tener el DBA completa jurisdicción de los datos se asegura que el único medio de accesar la base de datos sea a través de los canales establecidos, y que se definan controles de autorización que se aplican cada vez que se intenta el acceso a datos sensibles (aquellos datos cuyo valor es importante y que solo puede ser cambiado bajo condiciones muy particulares).

Integridad.
Se garantiza que los datos de la base de datos son exactos, esto es, no se presenta inconsistencia entre dos entradas que representan la misma cosa.

Independencia de los datos.
Para evitar que las aplicaciones dependan de la manera como los datos se organizan en el almacenamiento secundario y la manera como se accesan no sea dependiente de la aplicación. Cuando no existe independencia de datos es imposible cambiar la estructura de almacenamiento o la estrategia de acceso sin afectar la aplicación.

Modelos de Bases de Datos.
El modelo de datos es un grupo de herramientas conceptuales para describir datos, sus relaciones, su semántica y sus limitantes.
En general, los modelos de datos se pueden dividir en tres grupos:
Modelos lógicos basados en objetos
Su característica principal es que permiten estructuras flexibles y permiten especificar claramente las limitantes de los datos. Existen más de treinta modelos distintos de este tipo entre los cuales están el Binario, Semántico de Datos, Entidad-Relación, etc. El modelo Entidad-Relación es el más representativo de los modelos lógicos basados en objetos. Dicho modelo, se basa en un conjunto de

objetos básicos llamados entidades y de las relaciones entre estos objetos. Se manejan conceptos como Entidad y atributos.
Una Entidad es un objeto, persona, animal o cosa de interés para la comunidad de usuarios acerca de la cual es sistema de bases de datos debe mantener, correlacionar o desplegar información. Un atributo es una característica o cualidad de una entidad, debe ser de interés y estar dentro del alcance del sistema.
Lo que distingue a una entidad de otra es el conjunto de atributos que la describen. Por ejemplo los atributos nombre y dirección describen a un usuario especifico en una biblioteca.
Varias entidades pueden relacionarse entre sí mediante una relación o asociación, por ejemplo, la relación Libro-Autor asocia a un libro con cada uno de los autores que tiene (así pues, es posible tener un conjunto de entidades y un conjunto de relaciones, los cuales están constituidos por todas las entidades y todas las relaciones respectivamente).

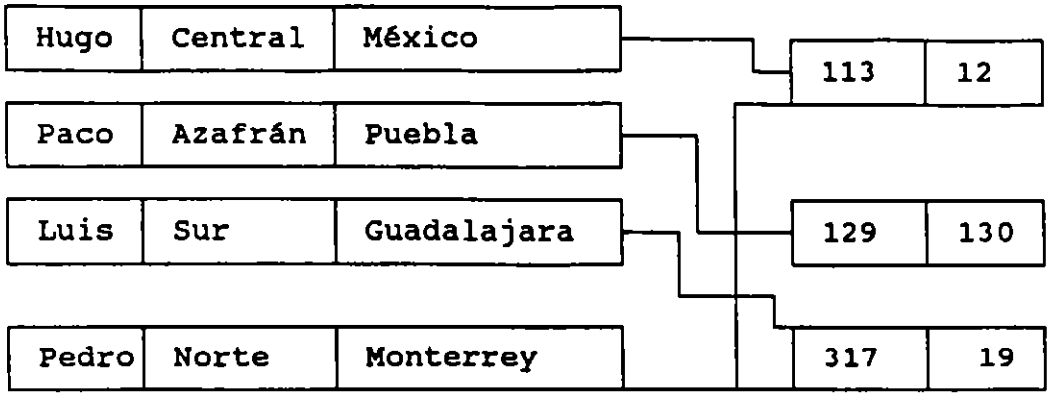
La estructura lógica general de una base de datos puede expresarse gráficamente por medio de un diagrama Entidad-Relación.
Modelos Lógicos Basados en Registros
A diferencia de los modelos de datos basados en objetos, estos especifican tanto la estructura lógica general de la base de datos como una descripción en un nivel mas alto de la implantación. Estos modelos, no permiten especificar en forma clara las limitantes de los datos. Los modelos de este tipo con mayor aceptación son:
Modelo Relacional
Modelo en el que los datos se representan por medio de una serie de tablas, cada una de las cuales tiene varias columnas con nombres únicos (atributos). Cada renglón de las tablas es llamado Registro o tupla; entonces, un registro representa un elemento de la tabla con diversos atributos.
La cardinalidad de una tabla queda definida como el número de registros (renglones) que tiene; y el orden de la tabla queda definido como el número de columnas o atributos de la misma.
Por ejemplo en la tabla B:

Tabla A				Tabla B	
nombre	calle	ciudad	número	número	cuota
Hugo	Central	México	113	113	12
Paco	Azafrán	Puebla	129	129	130
Luis	Sur	Guadalajara	317	317	19
Pedro	Norte	Monterrey	113		
Pedro	Norte	Monterrey	317		

se tiene tres tuplas o registros y dos columnas o atributos, por lo tanto la tabla tiene orden 2 y cardinalidad 3.

Modelos de Red.
Los datos en el modelo de Red se representan por medio de conjuntos de registros (en el sentido de la palabra en Pascal o PL/1) y las relaciones entre los datos se representan con ligas, que pueden considerarse como apuntadores. Los registros de la base de datos se organizan en forma de conjuntos de gráficas arbitrarias. La

siguiente figura es un ejemplo de una base de datos de red.



Modelos Físicos de los Datos
Sirven para describir los datos en un nivel más bajo. A diferencia de los modelos lógicos de los datos, son muy pocos los modelos físicos utilizados. Los más conocidos son:
El modelo unificador.
La memoria de cuadros.

TEMA 3. ESTIMACION DE COSTOS DEL SOFTWARE.

En los primeros días de la informática, el coste del software representaba un pequeño porcentaje del coste total del sistema informático, basado en computadora. Un error considerable en las estimaciones del coste del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro en muchos sistemas informáticos. Un gran error en la estimación del costo puede marcar la diferencia entre beneficios y pérdidas. Sobrepasar el coste puede ser desastroso para el equipo de desarrollo. La estimación del proyecto de software puede transformarse de un oscuro arte en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable.

Para realizar estimaciones seguras de coste y esfuerzo surge un número de opciones posibles:

1. Retrasar la estimación más adelante en el proyecto.

2. Utilizar "técnicas de descomposición" relativamente simples para generar las estimaciones del proyecto de software.

3. Desarrollar un modelo empírico para el coste y el esfuerzo del software.

4. Adquirir una o más herramientas automáticas de estimación.

Desafortunadamente, la primera opción, aunque atractiva, no es práctica: las estimaciones del coste deber ser proporcionadas "de antemano". Sin embargo, debemos reconocer que cuanto más tiempo esperamos, más cosas sabemos y, cuanto más sabemos, menor es la probabilidad de cometer serios errores en nuestras estimaciones.

Las tres opciones restantes son aproximaciones viables para la estimación del proyecto de software. Idealmente, las técnicas señaladas para cada opción deben ser aplicadas en conjunto, cada una usada para comprobar las otras. Las técnicas de descomposición utilizan una aproximación de "divide y vencerás" para la estimación del proyecto de software.

Los seres humanos han desarrollado una aproximación natural a la resolución de problemas: si el problema a resolver es demasiado complicado, tendemos a subdividirlo hasta encontrar problemas manejables. Entonces resolvemos cada uno individualmente y esperamos que las soluciones puedan ser combinadas para responder al problema original.

La estimación del proyecto de software es una forma de resolución de proyectos y, en la mayoría de los casos, el problema a resolver es demasiado complejo para considerarlo como una sola pieza. Por esta razón, descom-

ponemos el problema, re-caracterizándolo como un conjunto de pequeños problemas esperando que sean más manejables.

Estimaciones LDC y PF.

La medición es fundamental para cualquier disciplina de ingeniería del software no es una excepción.

Las métricas del software se refieren a un amplio rango de medidas para el software de computadoras. Dentro del contexto de la planificación del proyecto de software. Se considera al uso de líneas de código (LDC) como medidas clave. Los defensores de la medida LDC afirman que la LDC es un "artefacto" de todos los proyectos de desarrollo de software que puede ser fácilmente calculado, que muchos modelos de estimación del software existentes utilizan LDC o KLDC (miles de líneas de código) como clave de entrada y que ya existe un amplio conjunto de datos y de literatura basada en LDC.

Las métricas del software orientadas a la función son medidas indirectas del software y del proceso por el cual se desarrolla. Más que calcular las LDC, las métricas orientadas a la función se centran en la "funcionalidad" o "utilidad" del programa.

Los puntos de función son obtenidos utilizando una relación empírica basada en medidas contables del dominio de información del software y valoraciones subjetivas de la complejidad del software. La medida del punto de función fue diseñada originalmente para aplicarla en las aplicaciones de sistemas de información comerciales.

Elemento del dominio de informacion	Cuenta	Factor de ponderacion			PF
		Simple	Med.	Complejo	
Numero de entradas de usuario		3	4	6	
Número de salidas de usuario		4	5	7	
Número de peticiones de usuario		3	4	6	
Numero de archivos		7	10	15	
Numero de interfaces externas		5	7	10	

PF = cuenta por el factor de ponderación

(fig. 3.9). Los puntos de función son calculados rellenando la tabla que se muestra en la figura. Se determinan cinco características del dominio de la información y los cálculos aparecen indicados en la posición apropiada de la tabla. Los valores del dominio de la información están definidos de la siguiente manera:

Número de entradas de usuario: Se cuenta cada entrada de usuario que proporciona al software diferentes datos orientados a la aplicación. Las entradas deben ser distinguidas de las peticiones, que se contabilizan por separado.

Número de salidas de usuario: Se cuenta cada salida de usuario que proporciona al usuario información orientada a la aplicación. En este contexto, la salida se refiere a informes, pantalla, mensajes de error, etc. Los elementos de datos individuales dentro de un informe no se cuentan por separado.

Número de peticiones del usuario: Una petición está definida como una entrada interactiva que resulta en la generación de algún tipo de respuesta en forma de salida interactiva. Se cuenta cada petición distinta.

Número de archivos: Se cuenta cada archivo maestro lógico o sea, una agrupación lógica de datos que puede ser una parte de una gran base de datos o un archivo independiente.

Número de interfaces externas: Se cuentan todas las interfaces legibles por la máquina por ejemplo, archivos de datos en cinta o disco, que son utilizados para transmitir información a otro sistema.

Una vez obtenidos los datos mencionados, se deberá asociar un valor de complejidad a cada cuenta. Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada determinada es simple, medida o compleja.

Para calcular los puntos de función (PF) se utiliza la siguiente relación:

$$PF = cuenta.total \times [0.65 + 0.01 \times SUM(Fi)]$$

donde cuenta.total es la suma de todas las entradas obtenidas de la tabla. Los valores constantes de la ecuación anterior y los factores de ponderación aplicados a las cuentas de los dominios de información han sido determinados empíricamente.

Tema 4. Fundamentos del Diseño de Software

El diseño es el primer paso en la fase de desarrollo de cualquier sistema, se define como: "El proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física".

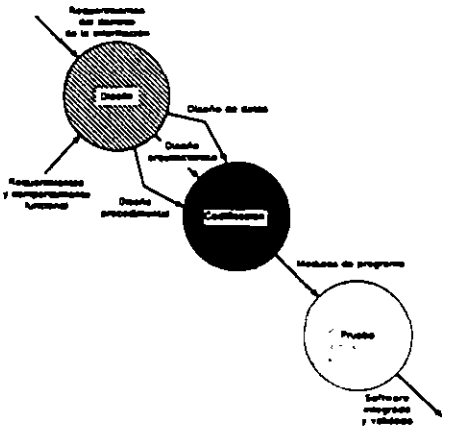
Hay que tener en cuenta que el objetivo del diseñador es producir un modelo de una entidad que será construida posteriormente. El proceso para el desarrollo de un modelo combina: intuición y criterios, es decir, se basa en la experiencia para construir entidades similares, principios que guían la forma en la que se desarrolla el modelo, criterios que facilitan discernir sobre la calidad y el proceso de representación del diseño final.

Subtema 4.1 La fase de desarrollo y el diseño de software

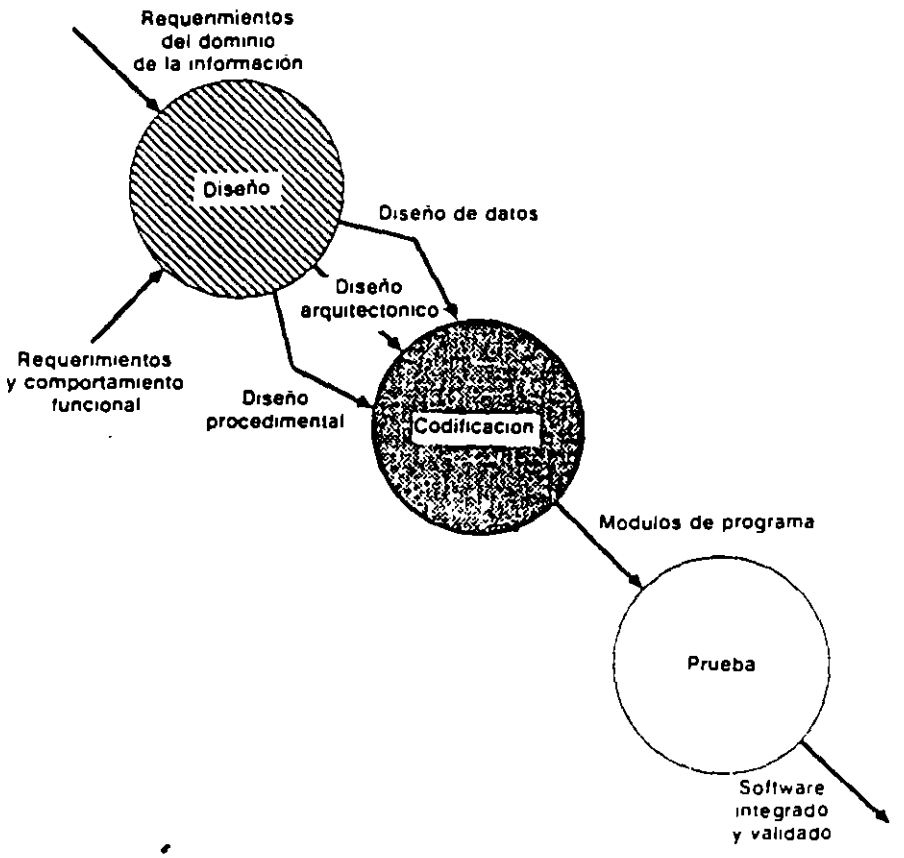
Una vez establecidos los requerimientos del software, la fase de desarrollo comprende tres pasos distintos:

1. Diseño
2. Generación de código
3. Prueba

Cada paso transforma la información de tal forma que finalmente se obtiene un software para computadora válido. En este tema explicaremos el primer paso "diseño" y posteriormente se desarrollaron los siguientes.



El flujo de información que alimenta la etapa de diseño se ve en la figura anterior En esta se aprecia las entradas las cuales son: los requerimientos del programa, los requerimientos funcionales y de comportamiento. Usando alguna de las distintas metodologias de diseño se realiza el diseño de datos, el diseño arquitectónico y el diseño procedimental, que posteriormente explicaremos.


Requerimientos
del dominio
de la información
Diseño
Diseño de datos
Diseño
arquitectonico
Requerimientos
y comportamiento
funcional
Diseño
procedimental
Codificacion
Modulos de programa
Prueba
Software
integrado
y validado

Cabe hacer notar que el diseño es el lugar en donde se asienta la calidad del desarrollo del programa, nos da las representaciones del software que pueden establecerse para conseguir un producto con calidad.

Subtema 4.2El proceso de diseño

El diseño de programas es un proceso mediante el cual se traducen los requerimientos en una representación del software.

Desde el punto de vista de una gestión de proyecto, el diseño del software se realiza en dos pasos:

1.Diseño preliminar: que se refiere a la transformación de los requerimientos en datos y arquitectura del software

2. Diseño en detalle: que se enfoca hacia los refinamientos de la representación arquitectónica que conduce a una estructura de datos detallada y a representaciones algoritmicas del software.

Hablaremos ahora acerca del diseño y calidad de software que es esencial para el logro de un buen producto.

Diseño y calidad del software

Para evaluar la calidad de una representación de diseño presentamos los siguientes criterios:

Un diseño debe:

- exhibir una organización jerárquica
(que haga uso inteligente de control entre los elementos del software)

- ser modular
(esto es, el software debe estar particionado lógicamente en elementos que realicen funciones y subfunciones especificas)

- contener una representación distinta y separada de los datos y los procedimientos.

- conducir a módulos que muestren características funcionales independientes

- derivarse usando un método repetitivo
(que esté conducido por la información obtenida durante el análisis de requerimientos de software)

Subtema 4.3 Fundamentos del diseño

Se ha mencionado ya la importancia que representa la elaboración de un buen diseño, para que esto se lleve a cabo de manera eficaz, vamos a tocar algunos aspectos fundamentales.

El establecimiento de conceptos fundamentales en el diseño del software, se ha dado desde hace tres décadas, los cuales ayudan al ingeniero en software a responder a las siguientes preguntas:

- Que criterios pueden usarse para subdividir el software en componente individuales?

- Como se separan los detalles de una función o una estructura de datos de una representación conceptual del software?

- Existen criterios uniformes que definen la calidad técnica de un diseño de programas?

Trataremos algunos de estos conceptos, sobre todo aquellos que respondan a las preguntas antes mencionadas. Comenzaremos con:

Refinamiento:

Es una temprana estrategia de diseño descendente propuesta por Niklaus Wirth, "En cada paso del refinamiento, una o varias instrucciones del programa dado se descomponen en más instrucciones detalladas. Esta descomposición sucesiva o refinamiento de especificaciones termina cuando todas las instrucciones están expresadas en términos del computador usado o lenguaje de programación".

El refinamiento es realmente un proceso de elaboración. Se comienza con una declaración de la función (o descripción de la información) que se define en el nivel superior de abstracción, es decir, la declaración describe la función o información conceptualmente, pero no se da información sobre el funcionamiento interno de la función o la estructura interna de la información, el refinamiento hace que el diseñador amplié la declaración original, dando mas y mas detalles conforme se produzcan sucesivos refinamientos.

Arquitectura de software

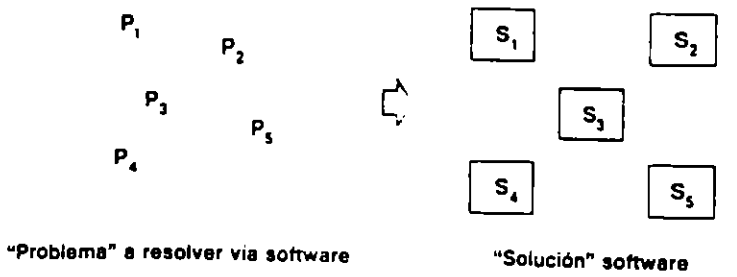
Otro concepto es la arquitectura del software y esto alude a dos características importantes en un programa:

1. la estructura jerárquica de los componentes procedimentales (módulos)

2. la estructura de datos

La arquitectura de software se deriva mediante un proceso de partición, que relaciona a los elementos de una solución del software, con partes de un problema real definido implícitamente durante el análisis de requerimientos. La evolución del software y la estructura

de datos comienza con una definición del problema. La solución ocurre cuando cada parte del problema se resuelve mediante uno o mas elementos de software, en la siguiente figura se muestra la representación de esta transición entre el análisis de requerimientos de software y el diseño.



Estructura de un programa

Un concepto mas es la estructura del programa que representa la organización jerarquica de las componentes del programa (módulos) e implica una jerarquía de control. Para representarla se utilizan notaciones diferentes, siendo la mas común el diagrama de árbol, llamado diagrama de estructura. En la figura siguiente se muestra un diagrama de árbol, con una profundidad y anchura que indican el numero de niveles de control y expansión global. Notese que el abanico de salida es una medida del numero de módulos que están directamente controlados por otros módulos, el abanico de entrada indica cuantos módulos controlan directamente a un modulo dado.

Estructura de datos

Siguiendo con los conceptos, uno mas es la representación de la relación lógica entre los elementos individuales de datos, conocidas como estructura de datos, la cual es muy importante y dicta la organización, los métodos de acceso, grado de asociatividad y alternativas de procesamiento para la información.

Algunas estructuras de datos clásicas, se muestran en la figura. El elemento escalar representa un elemento simple de información direccionado mediante un identificador. El vector es un conjunto continuo de elementos escalares que pueden ser de una, dos o más dimensiones. La lista enlazada esta organizada con elementos escalares, no continuos y pueden ser procesados como una lista. Cada nodo o elemento, contiene una organización propia y uno o más apuntadores que indican la dirección de memoria del siguiente nodo de la lista.

Procedimientos de software

El hablar de procedimiento de software, se enfoca sobre los detalles de procesamiento de cada módulo individualmente. El procedimiento debe dar una especificación precisa del procesamiento, incluyendo secuencia de sucesos, puntos de decisiones exactos, operaciones repetitivas e incluso organización/estructura de datos.

Existe una relación entre estructura y procedimiento, el procesamiento indicado por cada módulo debe incluir una referencia a todos los módulos subordinados del modulo que se describe.

Modularidad

Cuando se tiene un problema, para resolverlo, es más fácil dividirlo en partes manejables llamadas módulos, el modulo es un elemento con nombres y direcciones separadas, los cuales se integran para satisfacer los requerimientos del problema.

Es posible deducir que si subdividimos el software indefinidamente, el esfuerza que se requiere para desarrollar será significativamente pequeño. Sin embargo, hay otros factores que hacen que esta conclusión no sea del todo aceptada.

Abstracción

Al considerar una solución modular a cualquier problema, se pueden formular muchos niveles de abstracción, la cual facilita al diseñador representar un objeto de datos a diferentes niveles de detalle y especificarlo en el contexto de las operaciones que se le aplican. En el nivel superior de abstracción, se establece la solución en términos amplios usando el lenguaje del entorno del problema, en los niveles inferiores se toma una orientación más procedimental y se establece la solución en forma que pueda implementarse directamente.

Ocultamiento de la información

Al especificarse o diseñarse los módulos se debe hacer de tal manera que la información contenida dentro de un módulo sea inaccesible a otros módulos que no necesiten tal información.

Subtema 4.5 Diseño modular efectivo

En el subtema anterior tratamos acerca de la modularidad, la cual se ha convertido en un enfoque aceptado en todas las disciplinas de ingeniería. Algunas ventajas que proporciona el diseño modular es que reduce la complejidad, facilita los cambios y da como resultado una fácil implementación, posibilita además el desarrollo paralelo de diferentes partes de un sistema.

Tipos de módulos

Los módulos se dividen en categorías, las cuales son:

- módulo secuencial
(se llama y ejecuta sin interrupción aparente, son los más comunes y se categorizan como macros en tiempo de compilación y subprogramas convencionales, conocidas como funciones, procedimientos o subrutinas)

- módulo incremental
(se puede interrumpir antes de la terminación por software de aplicación y reestablecido en el punto de interrupción, se les llama comúnmente corrutinas, mantienen un apuntador de entrada que permite al módulo reestablecer el punto de interrupción)

- módulo paralelo
(se ejecuta simultáneamente con otro módulo en multiprocesamientos concurrentes, se encuentran en cálculos de alta velocidad que necesitan dos o mas procesadores trabajando en paralelo).

Independencia funcional

La comunicación entre módulos no debe ser excesiva y se deben desarrollar con una clara función de lo que va a hacer, para llegar a lo que se conoce como independencia funcional.

Siempre es preferible diseñar software de forma que cada módulo se enfoque a una subfunción específica de requerimientos y se tenga una interface sencilla.

Esta independencia se mide con dos criterios cualitativos:

- Cohesión

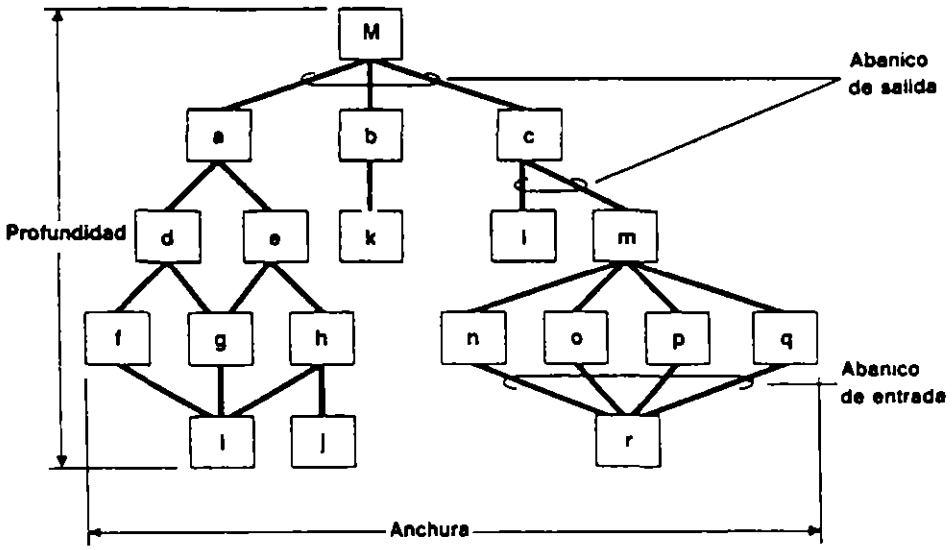
- Acoplamiento

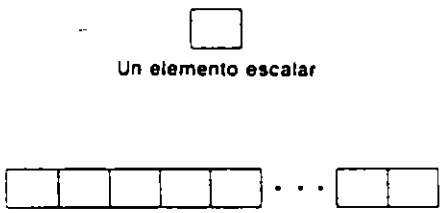
Cohesión:

Al hablar de un módulo coherente, se hace referencia a una tarea sencilla en un procedimiento de software y requiere poca interacción con procedimientos que ejecutan en otras partes de un programa, es decir, un módulo coherente debe hacer sólo una cosa.

La cohesión puede ser representada como el "espectro" mostrado en la siguiente figura, en la cual se nota los diferentes tipos de cohesión que se tienen de acuerdo al nivel.

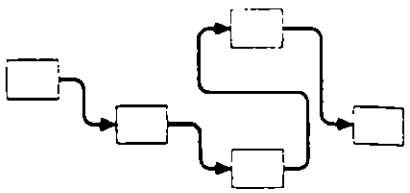
En la medida de la fortaleza funcional de un módulo, si se ejecutan un conjunto de tareas relacionadas con otras debilmente, se habla de **cohesión coincidental**, si se relacionan lógicamente es **cohesión lógica** y cuando se contienen tareas relacionadas por el hecho de que



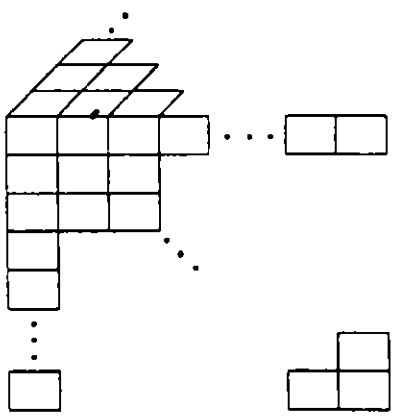


Un elemento escalar

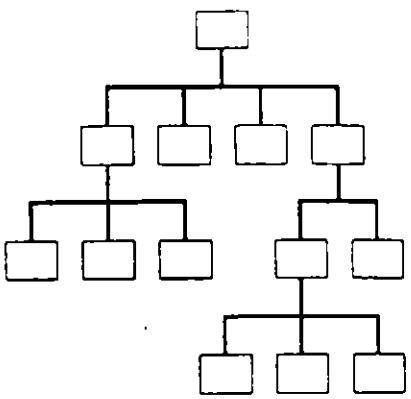
Un vector secuencial



Una lista enlazada

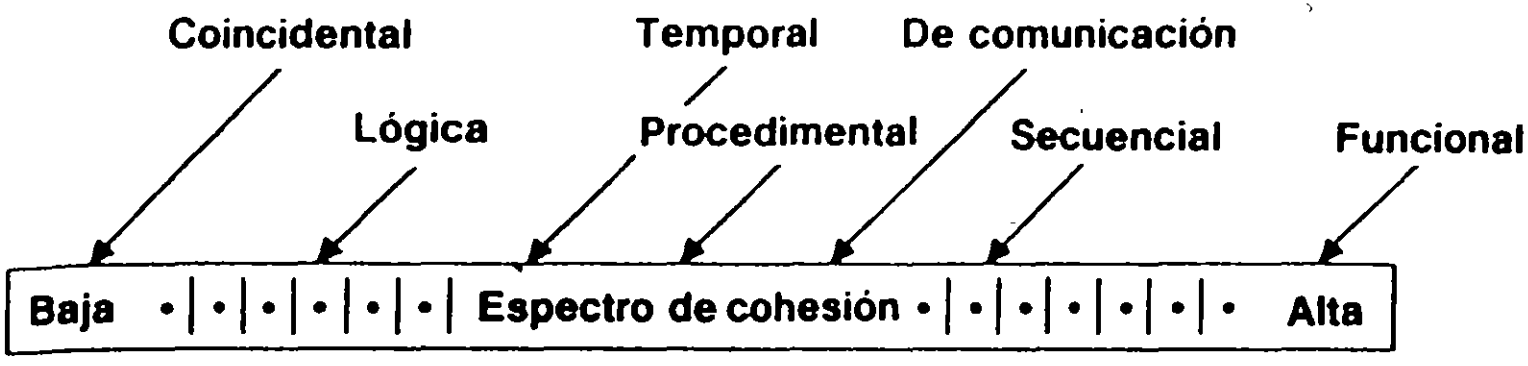


Un espacio *n*-dimensional

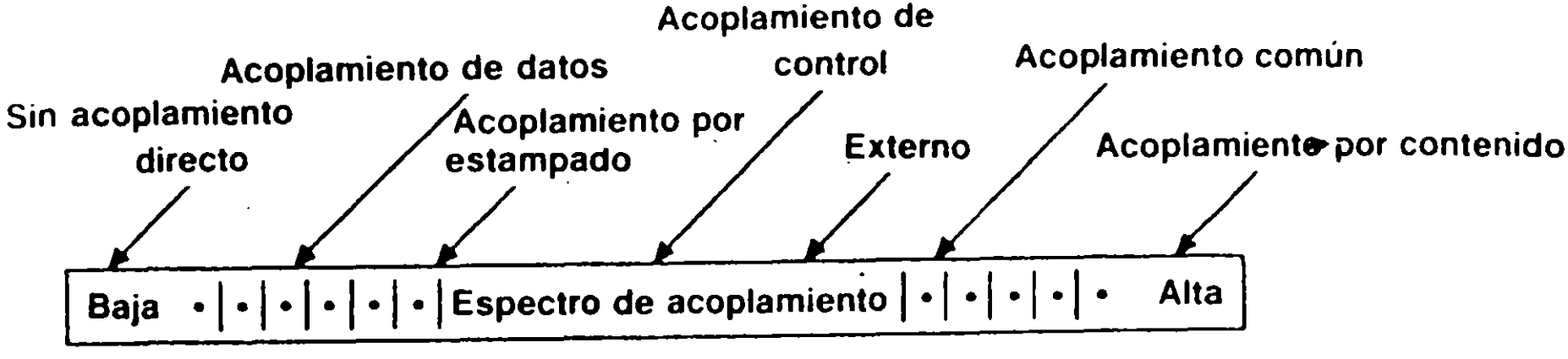


Un árbol jerárquico

Una medida de la fortaleza funcional relativa de un módulo



Una medida de la interdependencia entre módulos de software



se deben ejecutar dentro de un mismo tiempo, se exhibe **cohesión temporal**. Se puede apreciar también que la escala de la cohesión no es lineal, siempre se intenta conseguir un gran cohesión, aunque el punto medio del rango del espectro es aceptable.

Se habla de **cohesión procedimental**, cuando los elementos de procesamiento de un módulo están relacionados y deben ejecutarse en un orden especifico, y cuando los elementos de procesamiento se concentran sobre una área de un estructura de datos, se presenta **cohesión de comunicación.**

No es necesario determinar el nivel preciso de cohesión. Es importante buscar una cohesión alta y reconocer la cohesión baja de forma que el diseño del software pueda modificarse, de forma que contenga una mayor independencia funcional.

Acoplamiento

Otro criterio de medida de independencia, el cual es acoplamiento, que se puede definir como la medida de la interconexión entre módulos en una estructura de un programa, que depende de la complejidad de la interface entre módulos, del punto en el que se hace una entrada o referencia a un módulo y de los datos que pasan a través de la interface. Al igual que la cohesión se puede representar por un espectro, en el que se marcan los tipos de acoplamiento.

El **acoplamiento directo** se da cuando dos módulos están relacionados en forma directa, el **acoplamiento de datos** es cuando un modulo B es subordinado de un modulo A y esta accesible mediante una lista de argumentos convencionales, a través de los cuales pasa datos. Al pasar una porción de una estructura de datos mediante una interface de módulo que es una variación del acoplamiento de datos, se le conoce como **acoplamiento por estampado**. A niveles moderados, el acoplamiento se caracteriza por el paso de control entre módulos, en su forma sencilla, el control se pasa mediante un indicador sobre el que se toman las decisiones en un módulo subordinado o superior.

Se producen niveles altos de acoplamiento, cuando los módulos están ligados a un entorno externo al software. El **acoplamiento externo** es esencial, pero debe limitarse a un pequeño numero de módulos dentro de una estructura. Así mismo a niveles altos de acoplamiento se encuentra el **acoplamiento común**, que se da cuando varios módulos se hace referencia a un área de datos global. Y el mayor grado de acoplamiento se da en el **acoplamiento por contenido**, cuando un módulo hace uso de información de control o de datos mantenidos dentro de los limites de otro módulo.

Subtema 4.6 Diseño de datos

Una de las mas importantes de las tres actividades de diseño que se realizan durante la ingeniería de software, es el diseño de datos.

Esta actividad como ya mencionamos es la primera en el diseño de datos y se encarga principalmente de seleccionar las representaciones lógicas de los objetos de datos, conocidas

como estructuras de datos, las cuales se identifican durante las fases de definición y especificación de requerimientos.

Independientemente de las técnicas de diseño usadas, los datos bien diseñados nos pueden conducir a una mejor estructura del programa, modularidad y reducción de la complejidad de los procedimientos.

Al especificar los datos tenemos tener presentes los siguientes principios:

1. Los métodos de análisis sistemático, aplicados al software deben también aplicarse a los datos: deben desarrollarse y revisarse las representaciones de flujo y estructura de datos, considerar organizaciones alternativas y evaluar el impacto del diseño de datos sobre el diseño de software, es decir, si por ejemplo, la especificación de una estructura de lista doblemente ligada puede satisfacer agradablemente a los requerimientos de los datos, pero conducir a un diseño de software difícil de manejar.

2. Deben identificarse todas las estructuras de datos y operaciones que han de ejecutarse sobre cada una de ellas: Si por ejemplo, se considera una estructura de datos formado por un conjunto de diversos elementos de datos. La estructura será manipulada por varias funciones principales, para la evaluación de la operación ejecutada sobre esa estructura, se define un tipo abstracto de datos para usarlo en el diseño subsecuente del software, que puede llegar a simplificar el diseño de software.

3. Deben establecerse y usarse un diccionario de datos para definir el diseño de los datos y el software: En el diccionario de datos se representa explícitamente las relaciones entre los datos y las ligaduras de los elementos de una estructura de datos.

4. Las decisiones de diseño de datos a bajo nivel deben retrasarse hasta las ultimas etapas del proceso de diseño: Puede usarse un proceso de refinamiento sucesivo para el diseño de datos, es decir, definir una organización global de los datos durante el análisis de requerimientos, refinarse durante el diseño preliminar y especificarse en detalle durante el diseño detallado.

5. La representación de una estructura de datos debe ser conocida sólo por los módulos que hagan un uso directo de los datos contenidos dentro de la estructura: Este punto hace referencia a los conceptos antes vistos de ocultación de información y acoplamiento, los cuales proporcionan un aspecto importante en la calidad del diseño de software.

6. Debe desarrollarse una biblioteca de estructuras de datos útiles y de las operaciones que pueden aplicarse a ellas: Una biblioteca de estructuras de datos, pueden reducir el trabajo de especificación y diseño de los datos.

7. El diseño de software y el lenguaje de programación deben soportar la especificación y realización de tipos abstractos de datos: La implementación de una estructura de

datos sofisticada puede hacerse excesivamente difícil si no hay forma de realizar una especificación directa de la estructura.

Los principios descritos forman la base de un método de diseño de datos, que puede ser integrado en la fase de definición y desarrollo del proceso de ingeniería de software, dado que una clara definición de la información es esencial para desarrollar bien el software.

Subtema 4.7 Diseño Arquitectónico

El diseño principal del diseño arquitectonico es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos, además mezcla la estructura de programas y la estructura de datos, define las interfaces que facilitan el flujo de los datos a lo largo del programa.

Subtema 4.8 Diseño Procedimental

Una vez que se ha establecido la estructura del programa y de los datos, se realiza el diseño procedimental. En este se realiza la especificación procedimental que requiere definir los detalles algorítmicos que deben establecerse en un lenguaje natural, existe desafortunadamente un pequeño problema, en el diseño procedimental debe especificarse los detalles de los procedimientos sin ambigüedad y la falta de ambigüedad en un lenguaje natural no es corriente.

Programación Estructurada

En el desarrollo procedimental se requiere del uso de un lenguaje, como ya mencionamos, sus fundamentos se formaron a principios de los años 60, en el cual se propuso el uso de un conjunto de construcciones lógicas con las que podría formarse cualquier programa. Cada construcción tenía una estructura lógica predecible, se entraba por el principio y se salía por el final, y facilitaba al lector seguir más fácilmente el flujo procedimental. Las construcciones introducidas anteriormente en la representación de datos, son la secuencia, condición y repetición.

Secuencia: la secuencia, implementa los paso de procesamiento esenciales en la especificación de cualquier algoritmo.

Condición: la condición da la facilidad para seleccionar un procedimiento basado en alguna ocurrencia lógica.

Repetición: la repetición suministra el ciclo.

Estas tres construcciones son fundamentales en la programación estructurada, la cual es una técnica de diseño importante en el campo más amplio de lo que hemos aprendido de ingeniería del software.

Cualquier programa, independientemente del área de aplicación o complejidad técnica, puede diseñarse e implementarse usando sólo las tres construcciones estructuradas.

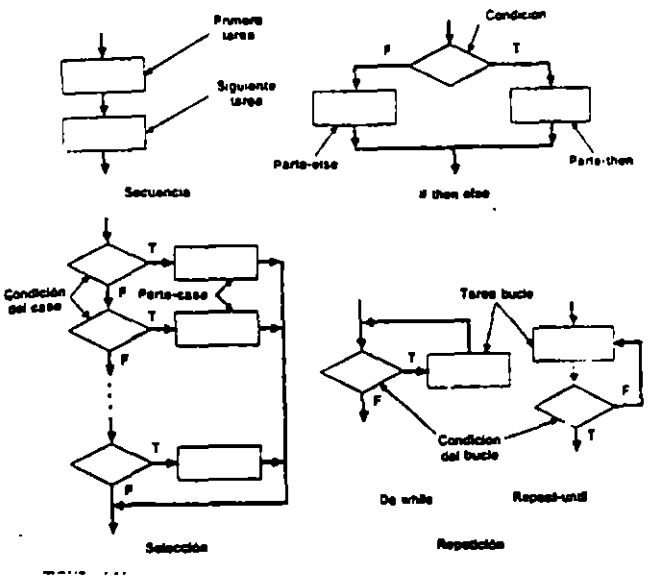
Herramientas gráficas de diseño

Alguien alguna vez dijo: un dibujo vale mas que mil palabras, pero es importante saber que dibujo y que miles de palabras"... No hay duda que las herramientas gráficas, tales como los diagramas de flujo o de cajas, dan una excelente forma gráfica de describir fácilmente los detalles procedimentales.

Diagrama de flujo:

El diagrama de flujo es la representación gráfica más usada para el diseño procedimental. Pero también es del que mas se ha abusado. Es un gráfico muy sencillo, una caja indica un paso de procesamiento, un rombo representa una condición lógica y las flechas muestran el flujo de control.

La siguiente figura muestra las tres construcciones de programación estructurada.



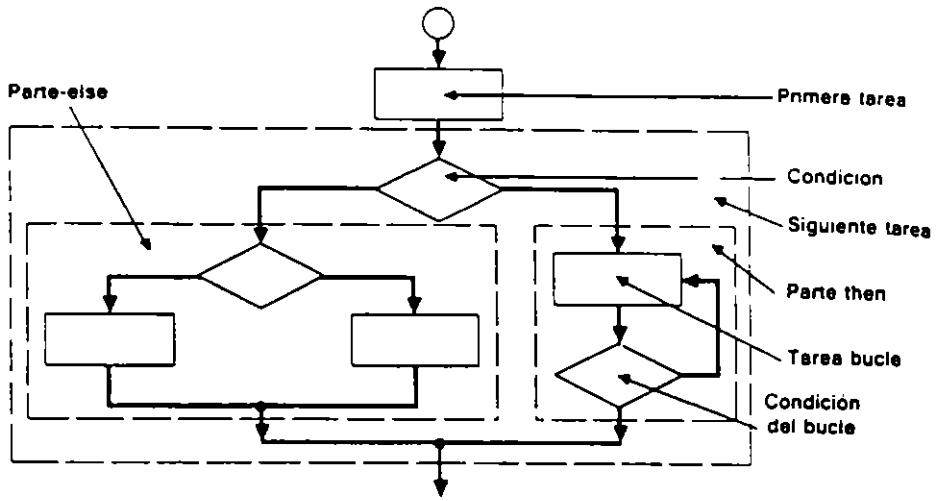
La secuencia esta representada por dos cajas de procesamiento conectadas por una línea de control.

La condición también llamada IF-THE-ELSE, se dibuja como un rombo de decisión, el cual, si es verdad, hace que se realice el procesamiento de la parte then y si es falso llama al procesamiento de la parte del ELSE.

La repetición se representa usando dos formas algo diferentes. La forma DO-WHILE prueba una condición y ejecuta repetidamente una tarea mientras la condición sea verdadera. La

forma REPEAT-UNTIL ejecuta primero el ciclo, luego checa la condición y repite la tarea hasta que falla la condición.

Estas construcciones estructuradas pueden estar anidadas unas en otras como se muestra en la siguiente figura.



Una herramienta mas en el diseño gráfico es el diagrama de cajas, que surgió del deseo procedimental que no permitiera la violación de las construcciones estructuradas, y tiene las siguientes características:

- Dominio funcional:
el ámbito de la repetición o un IF-THEN-ELSE, esta bien definido y claramente visible como una representación gráfica.

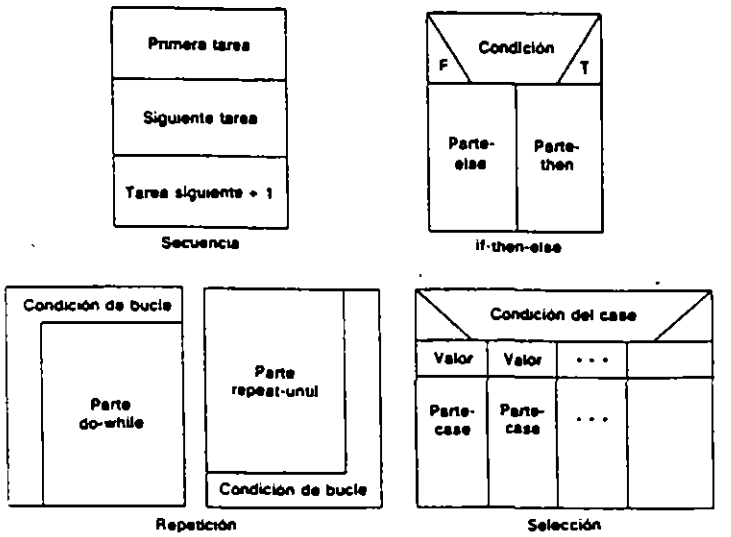
- Imposible transferencia arbitraria de control

- Determinación de los ámbitos de los datos locales y/o globales, estos pueden determinarse fácilmente.

- Representación fácil de la recursividad.

En la siguiente figura se muestra, la representación gráfica de construcciones estructuradas usando diagramas de cajas, en esta se ve que el elemento fundamental del diagrama es la caja, para representar la secuencia se conectan dos cajas seguidas. para representar el IF-THEN-ELSE, se pone una caja de la parte THEN y otra de la parte ELSE, a continuación de la caja de condición. La repetición se dibuja con una caja interior que encierra el proceso que ha de repetirse (parte del DO-WHILE o parte REPEAT-UNTIL).

Y la selección se representa usando la forma gráfica mostrada al final.



Las llamadas a los módulos subordinados puede representarse mediante una caja con el nombre del módulo encerrado dentro de un óvalo.

Lenguaje de diseño de programas

Para el diseño de programas se requiere de un lenguaje, llamado pseudocódigo, que es un lenguaje mezclado que utiliza el vocabulario de un lenguaje por ejemplo el inglés, y la sintaxis general de otro por ejemplo, un lenguaje de programación estructurada.

La diferencia entre el lenguaje de diseño de programas y un lenguaje de programación de alto nivel real se refiere al uso del texto narrativo insertado directamente dentro de las sentencias del lenguaje de diseño de programas.

Independientemente de su origen, un lenguaje de diseño debe tener las siguientes características:

- Sintaxis fija de palabras clave que den todas las construcciones estructuradas, declaraciones de datos y características de modularidad

- Sintaxis libre en lenguaje natural que describa las características del procesamiento

- Facilidad para declaración de datos que incluyan las estructuras de datos sencillas (escalares y arreglos) y complejas (listas ligadas o árboles).

- Definición de subprogramas y técnicas de llamada que soporten los distintos modos de descripción de interfaces.

La sintaxis básica del LDP debe incluir:

- definición de subprograma

- descripción de interfases

- declaración y tipificación de datos

- técnicas para estructurar en bloques

- instrucciones de condición

- instrucciones repetitivas

- instrucciones de entrada/salida.

El formato y semántica de un LDP es:

- Instrucción que permite representar las estructuras de datos locales y globales:

TYPE [nombre-variable] IS [calificador1] [calificador2]

donde:

[nombre-varible]	es una varible declarada dentro de un módulo o declarada para uso global o entre módulos.
[calificador1]	indica la estructura de datos especifica e incluye palabras reservadas como: SCALAR, ARRAY, LIST, STRING, STRUCTURE
[calificador2]	indica como se utilizan los nombres de variables en el contexto de un módulo o programa.

- Elementos procedimentales estructurados en bloques, es decir, pude definirse un pseudocódigo en bloques que se ejecute como una entidad simple.

Un bloque esta delimitado de la siguiente manera:

```
BEGIN [nombre-bloque]

        [sentencias de pseudocódigo];

END
```

donde, [nombre-bloque] puede usarse para dar un nodo de referencia subsecuente a un bloque y [sentencias en pseudocódigo] son una combinación de otras instrucciones en LDP.

- Instrucción de condición en LDP, tiene la forma clásica if-then-else.

```
IF [descripción-condición]
    THEN [bloque o sentencia de pseudocódigo];
    ELSE [bloque o sentencia de pseudocódigo];
ENDIF
```

donde, [descripción-condición] indica la decisión lógica que debe hacerse para llamar a la parte de procesamiento THEN o ELSE.

- Instrucción de selección, es un conjunto de IF anidados, esta instrucción compara un parámetro especifico, la variable del case, con un conjunto de condiciones. Cuando se satisface una condición, se llama a un bloque o sentencia individual en pseudocódigo. Su representación es:

```
CASE OF [nombre-variable-case]
    WHEN [condición1-case] SELECT [bloque o sentencia de pseudocódigo];
    WHEN [condición2-case] SELECT [bloque o sentencia de pseudocódigo];
    ....
    WHEN [ultima-condición-case] SELECT [bloque o sentencia de pseudocódigo];
    DEFAULT [fallo o error del case. bloque o sentencia de pseudocódigo];
    ENDCASE
```

- Instrucción de repetición en LDP, incluye ciclos pre-test y post-test así como un ciclo con índice.

```
DO WHILE descripción-condición
    bloque o sentencia en pseudocódigo
ENDDO
```

REPEAT UNTIL descripción-condición

bloque o sentencia en pseudocódigo

ENDREP

DO FOR [índice = lista-índice, expresión o secuencia]

bloque o sentencia en pseudocódigo

ENDFOR

- Los subprogramas y sus correspondientes interfases se definen usando las siguientes instrucciones del LDP.

PROCEDURE [nombre-subprograma][atributos]

INTERFACE [lista-argumentos]

[bloques y/o sentencias en pseudocódigo];

donde, [atributos] de un subprograma describe sus características de referencia (internas o externas) y otros atributos dependientes de la implementación (lenguaje de programación) si los hubiera INTERFACE se utiliza para especificar una lista de argumentos del módulo que contiene los identificadores para la información de llegada y salida.

- Especificaciones de entrada/salida, dependen mucho de los lenguajes de diseño, las mas comunes son:

READ/WRITE TO [dispositivo] [lista E/S]

donde dispositivo indica el dispositivo físico de E/S y lista E/S contiene las variables que se transmiten.

Comparación de notaciones de diseño

Cualquier notación para el diseño procedimental, si se usa correctamente, puede ser de ayuda muy valiosa en el proceso de diseño, es por esto que cualquier comparación debe basarse en esta premisa, recíprocamente, incluso la mejor notación, si se aplica mal, añade poco a la compresión. Tomando en consideración lo anterior podemos aplicar comparaciones a las notaciones:

Modularidad:

Una notación de diseño debe soportar el desarrollo de software modular.

Simplicidad global:

Una notación de diseño debe ser relativamente sencilla de aprender, relativamente fácil de usar y generalmente fácil de leer.

Facilidad de edición:

La facilidad con la que una representación de diseño pueda ser editada, puede ayudar a facilitar cada uno de estos pasos de la ingeniería de software.

Legible por la máquina:

Una notación que pueda ser introducida directamente en un sistema de desarrollo basado en computadora, ofrece enormes beneficios potenciales.

Mantenimiento:

Es esta la fase mas costosa del ciclo de vida del software. El mantenimiento de la configuración de software casi siempre significa mantenimiento de la representación del diseño procedimental.

Exigencia de estructura:

La notación de diseño que refuerce el uso de únicamente construcciones estructuradas, promueve la práctica de un buen diseño.

Representación de datos:

La habilidad para representar los datos locales y globales es un elemento esencial en el diseño detallado. Una notación puede idealmente, representar directamente los datos.

Verificación lógica:

La verificación automática de la lógica del diseño es un objetivo supremo durante la prueba de software. Una notación que la refuerce mejora grandemente la suficiencia de la prueba.

Disposición de la codificación:

Una notación que se convierta fácilmente a código fuente reduce el trabajo y los errores.

Parece que un lenguaje de diseño de programas ofrece la mejor combinación de características. Se puede insertar directamente en los listados fuente, mejorando de esta forma la documentación y haciendo menos difícil el mantenimiento del diseño.

Tema 5 DISEÑO ORIENTADO AL FLUJO DE DATOS.

El diseño se ha descrito como un proceso multipaso en el que las representaciones de la estructura de datos, estructura de programa y procedimiento, se sintetizan a partir de los requerimientos de la información. Es además, una actividad referida a la toma de decisiones importantes, frecuentemente de una naturaleza estructural. Comparte con la programación los aspectos referentes a la abstracción en la representación de la información y en las secuencias de procesamiento, pero el nivel de detalles es muy diferente en ambos casos. El diseño construye representaciones coherentes y bien planificadas de programas, concentrándose en las interrelaciones de las partes a un nivel más alto y en las operaciones lógicas implicadas en los niveles inferiores.

En este tema trataremos acerca del diseño orientado al flujo de datos (y en general el diseño de software), el cual tiene sus orígenes en los primeros conceptos de diseño que aventuraron la modularidad, diseño descendente y programación estructurada. Sin embargo, el enfoque de diseño orientado al flujo de datos amplió estas técnicas procedimentales, integrando explícitamente el flujo de la información en el proceso de diseño.

Subtema 5.1 Areas de aplicación.

El diseño orientado al flujo de datos puede aplicarse a un amplio rango de áreas de aplicación. Por lo cual es un factor importante de selección de un método de diseño, la amplitud de las áreas en la que se aplica.

Un método de diseño orientado al flujo de datos es particularmente útil cuando la información se procesa secuencialmente y no existe una estructura de datos jerárquica.

Las técnicas de diseño orientadas al flujo de datos se aplican también a aquellas que se relacionen con el procesamiento de datos y pueden aplicarse con efectividad, incluso cuando existen estructuras de datos jerárquicas.

Subtema 5.2 Consideraciones sobre el proceso de diseño.

El diseño orientado al flujo de datos permite una cómoda transición de las representaciones de la información a una descripción de diseño de la estructura del programa.

Esta transición se realiza en cinco pasos:

1) se establece el tipo del flujo de información;

2) se indican los límites del flujo;

3) el diseño de flujo de datos se convierte en la estructura del programa;

4) se define la jerarquía de control mediante factorización, y

5) se refina la estructura resultante usando medidas y heurísticas de diseño.

Subtema 5.3 Flujo de transformación y Flujo de Transacción.

Analizaremos ahora dos tipos de flujo de información. Primero veremos el flujo de transformación:

Recordando el modelo fundamental del sistema (diagrama de flujo de datos), la información debe entrar y salir en una forma del "mundo externo". Por ejemplo, los datos escritos sobre un teclado de una terminal, hablados sobre una línea de teléfono y dibujados sobre una presentación gráfica por computadora, son todos formas de información del mundo externo. Tales datos externos deben ser convertidos en una forma interna para el procesamiento.

La historia del tiempo del flujo de datos se muestra en la siguiente figura, en la que se aprecia, la información entra al sistema mediante caminos que transforman los datos externos en una forma interna y se identifica como flujo de llegada. En el núcleo del software ocurre una transición. Los datos de llegada se pasan a través del centro de transformación y comienzan a moverse a lo largo de caminos que conducen ahora hacia la "salida" del software, se llaman flujo de salida.

En lo que se refiere al flujo de transacción tenemos que :

El modelo fundamental del sistema implica un flujo de transformación; el flujo de información se caracteriza frecuentemente por un elemento de datos sencillo, llamado una transacción, que desencadena otro flujo de datos a lo largo de uno de los muchos caminos. Cuando un DFD toma la forma mostrada a continuación, se presenta un flujo de transacción.

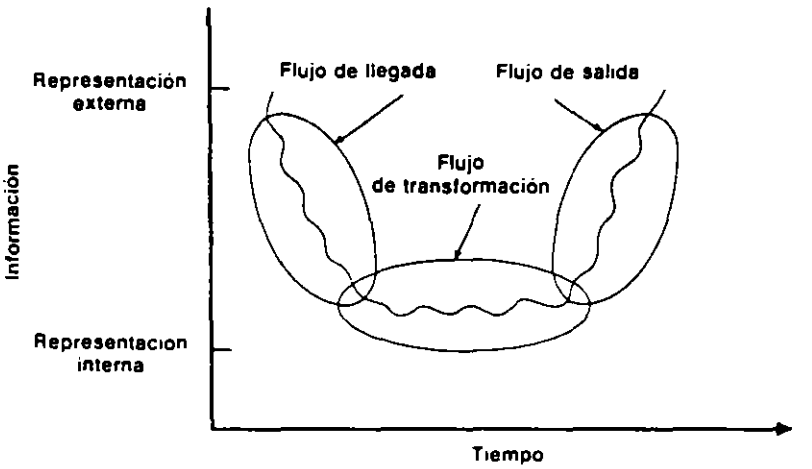
Este flujo de transacción se caracteriza por datos que se mueven a lo largo de un camino de llegada que convierte información del mundo externo en una transacción, esta transacción es evaluada, y basándose en este valor el flujo se inicia por uno de los muchos caminos de acción. El centro de flujo de información desde que salen muchos caminos de acción se llama un centro de transición. En un DFD para un gran sistema, pueden presentarse los dos tipos de flujos de transformación y de transacción.

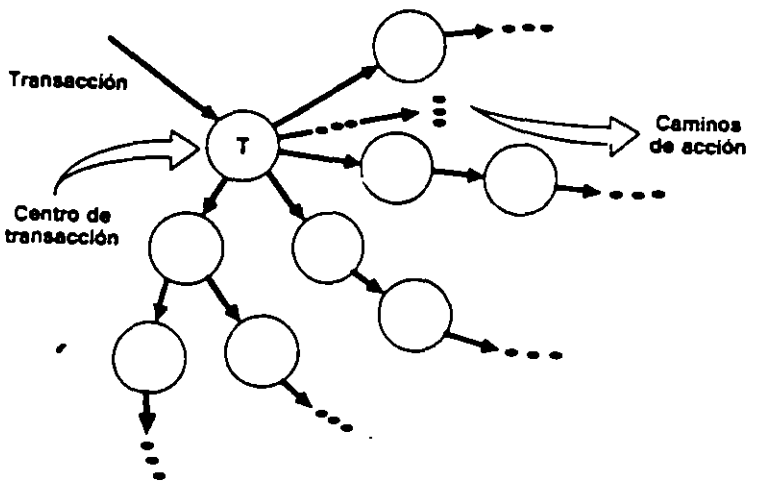
Para completar este punto vamos a definir que es un Análisis de Transformación.

El análisis de transformación es un conjunto de pasos de diseño que permiten a un DFD, con características de flujo de transformación, convertirse en una templeta predefinida para la estructura del programa.

Subtema 5.4 Pasos del diseño.

Los pasos de diseño comienzan con una revaluación del trabajo hecho durante el análisis de requerimientos y luego se va hacia el desarrollo de la estructura del programa. Analicemos paso a paso como se lleva a cabo este diseño:


Representación externa
Flujo de llegada
Flujo de salida
Flujo de transformación
Información
Representacion interna
Tiempo


Transacción
T
Caminos de acción
Centro de transacción

Paso 1. Revisión del modelo fundamental del sistema. El modo fundamental del sistema abarca el nivel 01 DFD y el soporte de la información. En realidad, el paso de diseño comienza con una evaluación de la Especificación del Sistema y de la Especificación de los Requerimientos del Software. Ambos documentos describen el flujo y estructura de la información de la interfase del software.

Paso 2. Revisión y refinamiento de los diagramas de flujo de datos para el Software. Usando la información obtenida de una Especificación de Requerimientos del software, se deriva un diagrama de flujos de datos a nivel 03 para el implicado en una transformación ejecuta una función sencilla y diferenciada, el DFD contiene los suficientes detalles para un "primer corte" en el diseño de la estructura de programa, y procedemos sin más refinamientos.

Paso 3. Determinar si el DFD tiene características de transformación de transacción, el flujo de información de un sistema puede presentarse siempre como transformación. Sin embargo, cuando se encuentra una transacción, se recomienda una organización de diseño diferente.
En este paso, el diseñador selecciona una característica global del flujo basándose en la naturaleza prevalente del DFD. Además, se aíslan las regiones locales del flujo de transformación o transacción. Estos subflujos pueden usarse para refinar la estructura del programa derivada de la característica global descrita anteriormente.

Paso 4. Aislar el centro de transformación especificando los límites del flujo de llegada y salida.
Los límites del flujo de llegada y salida están abiertos a diferentes diseñadores pueden seleccionar puntos algo diferentes en el flujo como posiciones límites, pueden derivarse soluciones de diseño alternativas variando la colocación de los límites del flujo.

Paso 5. Realización del "primer nivel de factorización". La estructura del programa representa una distribución descendente del control. La factorización da como resultado una estructura de programa en la que los módulos de nivel superior toman las decisiones de ejecución y los módulos de nivel inferior ejecutan la mayoría del trabajo de entrada, computacional y de salida. Los módulos de nivel intermedio ejecutan algún control y realizan moderadas cantidades de trabajo.

Primer nivel de factorización.
El número de módulos del primer nivel debe limitarse al mínimo necesario, para que puedan realizarse las funciones de control y mantener al mismo tiempo unas buenas características de acoplamiento y cohesión.

Paso 6. Ejecución del "segundo nivel de factorización". El segundo nivel de factorización se realiza mediante la conversión de las transformaciones individuales de un DFD, en los módulos correspondientes de la estructura del programa. Comenzando en el límite del centro de transformación y yendo hacia fuera a lo largo de los

caminos de llegada y luego de salida, las transformaciones se convierten en niveles subordinados de la estructura del software.

Una estructura de programa de primer corte puede siempre refinarse aplicando los conceptos de independencia de módulos. Se puede aumentar o reducir el número de módulos para producir una estructura que pueda implementarse sin dificultad, probarse sin confusión y mantenerse sin problemas.

El objetivo de los siete pasos procedentes es desarrollar una representación global del software. Esto es, una vez que se define la estructura, podemos evaluar y refinar la arquitectura del software viéndola como un todo.

Subtema 5.5 ANALISIS DE TRANSACCION

En muchas aplicaciones software, un elemento de dato determina uno o más flujos de información que afectan a la función implicada por el elemento de dato determinante. El elemento de dato, llamado transacción.

Pasos del diseño

Los pasos del diseño para el análisis de transacciones son similares y en algunos casos idénticos a los pasos para el análisis de transformaciones. La principal diferencia se refiere a la conversión del DFD en estructura de programa.

Paso 1. Revisar el modelo fundamental del sistema.

Paso 2.Revisar y refinar los diagramas de flujo de datos par software.

Paso 3.Determinar si el DFD tiene características de transformación o de transacción. Deben establecerse límites de flujo para ambos tipos de flujo.

Paso 4.Identificar el centro de transacción y las características del flujo de cada camino de acción. La posición del centro de transacción puede discernirse inmediatamente a partir del DFD. El centro de transacción está ligado al origen de varios caminos de información que fluyen radialmente de el.

Paso 5.Transformar el DFD en una estructura software adecuada al procesamiento de transacciones. El flujo de transacción se convierte en una estructura de programa que contiene una bifurcación de entrada y una bifurcación de salida o expedición. La estructura para la bifurcación de entrada se desarrolla de la misma forma que el análisis de transformaciones.

Paso 6.Factorizar y refinar la estructura de transacciones y la estructura de cada camino de acción. Cada camino de acción del diagrama de flujo de datos tiene sus propias características de flujo de información, se puede encontrar un flujo de transformación de transacción.

La transformación crear y visualizar mensaje del DFD se convierte en un módulo de utilidad (es decir, un módulo llamado por dos o más módulos) que se utiliza por dos estructuras de flujo de acción.

Paso 7.Refinar la estructura del software de "primer corte" usando medidas y heurísticas de diseño. Criterios tales como independencia de módulos, práctica y mantenimiento, deben ser considerados cuidadosamente cuando se propagan modificaciones a la estructura.

Una vez que se ha desarrollado la estructura del programa usando el diseño orientado al flujo de datos, puede conseguirse una modularidad efectiva aplicando los conceptos introducidos en el tema anterior y manipulando la estructura resultante de acuerdo al siguiente conjunto de criterios.

I.Evaluar la estructura de programa preliminar para reducir el acoplamiento y mejorar la cohesión. Los módulos pueden expandirse o reducirse con la mirada puesta en la mejora de la independencia de los módulos.

II.Intentar minimizar las estructuras con un abanico de salida ancho; fomentar los abanicos de entrada conforme se incrementa la profundidad.

III.Mantener el efecto de un módulo dentro del ámbito de control de ese módulo.

IV.Evaluar las interfases de los módulos para reducir la complejidad y redundancia y mejorar la consistencia.

V. Definir módulos cuyas funciones sean predecibles, pero evitar módulos que sean demasiado restrictivos.

VI.Buscar los módulos de una única entrada y una única salida, evitando las conexiones patológicas.

VII.Empaquetar el software basándose en las restricciones del diseño y requerimientos de transportabilidad.

En forma global, podemos decir que las técnicas expuestas en este tema conducen a una descripción del diseño preliminar del software. Se definen módulos, se establecen las interfases y se desarrollan las estructuras de datos. Estas representaciones del diseño forman la base de todo el trabajo posterior de desarrollo.

Tema 6 Diseño Orientado a las Estructuras de Datos

En este tema trataremos la relación entre el software y datos, la cual siempre ha existido desde los orígenes de la computación, comenzaremos diciendo que la estructura de la información llamada estructura de datos, es un factor que tiene impacto en la complejidad y eficiencia de los algoritmos diseñados para procesar la información.

La identificación de las estructuras de datos inherentes es vital y la estructura de los datos puede usarse para derivar la estructura de un programa.

Los datos de entrada, internamente información almacenada y los datos de salida, pueden tener cada uno una estructura única. El diseño orientado a la estructura de datos hace uso de estas estructuras como base para el desarrollo de software.

Nos daremos cuenta a lo largo de este tema que las estructuras de datos son tan importantes dado que afectan al diseño tanto en el aspecto estructural, como procedimental del software. Transforma además una representación de la estructura de datos en una representación de software, los creadores del diseño orientado a la estructura de datos han de definir un conjunto de procedimientos de transformación que utilizan la estructura de la información(datos) como guía.

Subtema 6.1 Técnicas de estructura de datos frente a las de flujo de datos

Antes de considerar las diferencias entre el diseño orientado al flujo de datos y orientado a la estructura de datos, es importante hacer mención que ambos comienzan en los paso de análisis que conducen a los siguientes pasos de diseño. Ambos son conducidos por la información; ambos intentan transformar la información en una representación de software.

El diseño orientado a la estructura de datos no hace un uso explícito de un diagrama de flujo de datos. El objetivo último de los métodos orientados a la estructura de datos es producir una descripción procedimental del software. El concepto de estructura modular de programa no se considera explícitamente. El diseño orientado a la estructura de datos hace uso de un diagrama jerárquico para representar la estructura de la información.

Subtema 6.2 Consideraciones sobre el proceso de diseño

El análisis de requerimientos del software permanece como la base del diseño orientado a la estructura de datos. La descripción del dominio de la información (estructura, contenido y flujo de datos) contenida en la Especificación de Requerimientos de Software prefigura la arquitectura del software que ha de desarrollarse durante del diseño.

Los métodos orientados a la estructura de datos tienen su propio conjunto de reglas. Sin embargo, las tareas de diseño que deben siempre realizarse son:

1.- evaluar las características de la estructura de datos

2.- representar los datos en términos de formas elementales, tales como secuencia, selección y repetición.

3.- transformar la representación de la estructura de datos en una jerarquía de control para el software

4.- refinar la jerarquía del software usando los criterios definidos como parte de un método

5.- desarrollar finalmente una descripción procedimental del software.

Subtema 6.3 Descomposición usando estructuras de datos.

Los orígenes del diseño orientado a la estructura de datos pueden dirigirse a las dos técnicas de descomposición usando estructuras de datos desarrolladas en forma independiente y prácticamente al mismo tiempo por Jackson en Inglaterra y Warnier en Francia.

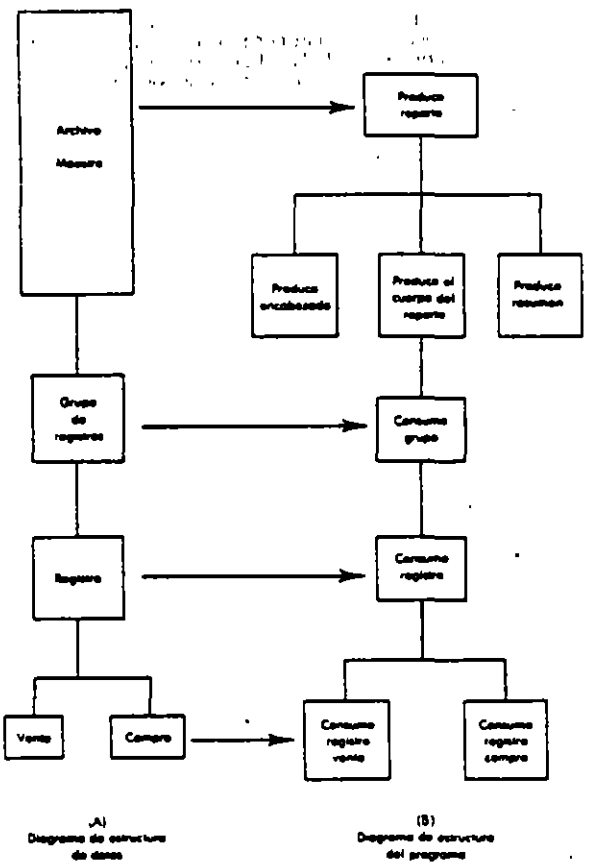
Estas técnicas están basadas en el principio de correspondencia entre la estructura del programa y la del problema que la programación pretende resolver. Como generalmente el problema a resolver consiste en el procesamiento de datos, la estructura del programa deberá corresponder a la estructura de los datos que procesa. Jackson menciona : "si un programa no corresponde al ambiente del problema, el programa no es ni pobre ni malo: es incorrecto".

Para comprender mejor esto, consideremos el siguiente problema:

En un almacén de una fábrica se requiere producir un reporte que presente el movimiento neto de productos (artículos, piezas, partes, etc), que el almacén compra y vende. Se cuenta con un sistema de cómputo que tiene un archivo maestro de transacciones, en donde se registra cada venta y cada compra. Los registros del archivo contienen, entre otras variables, el número de identificador del artículo, el tipo de movimiento (venta o compra) y la cantidad. El archivo se encuentra clasificado por número de artículo, de tal forma, que todos los registros que representan los movimientos que ha tenido un artículo se encuentran secuencialmente ligados.

El programa deberá producir el movimiento neto de cada artículo y al final resumirá el total de productos que sufrieron movimientos. El reporte final deberá tener el formato como se muestra en la figura.... ,

La estructura del programa deberá derivarse de la estructura de datos. Puesto que se desea obtener una estructura jerárquica, los datos serán analizados jerárquicamente. El diagrama de estructura de los datos, así como el diagrama de estructura de programa, representan la relación "es compuesto de". Por ejemplo, como entrada al programa tenemos el archivo de transacciones que "es compuesto de " un grupo de registros y cada grupo de registros "es compuesto de " registros individuales y, finalmente, cada registro es de uno de dos tipos: venta o compra, en el diagrama siguiente se puede apreciar lo anterior...



Por otro lado, a la salida del programa el reporte "es compuesto de " un encabezado, un cuerpo (todos los movimientos netos) y el resumen.

La estructura del programa se deriva de la estructura del reporte y de la de los datos de entrada al programa: el programa que produce el reporte "es compuesto de " un módulo que produce el encabezado, de un módulo que produce el cuerpo y de uno que produce el resumen. A su vez, el módulo que produce el cuerpo contiene un módulo que procesa un grupo de registros, requiriendo al mismo tiempo, de un módulo que emplea un registro y finalmente, dado que los registros son de dos tipos, se necesitaran dos diferentes módulos: uno para analizar los de tipo de venta y otro para consumir los de tipo compra, como se aprecia en la figura.

Como puede apreciar, una parte del diagrama de estructura del programa que produce el reporte corresponde a la estructura de los datos de entrada (el archivo de transacciones), y otra parte corresponde a la estructura de los datos de salida (el reporte).

Subtema 6.4 Areas de aplicación

El diseño orientado a la estructura de datos puede aplicarse con éxito a aplicaciones que tengan una estructura jerárquica, bien definida, de la información. Algunas son:

- Aplicaciones en sistemas de información comerciales.

La entrada y salida tiene distinta estructura (como en el ejemplo anterior), el uso de una base de datos jerárquica es frecuente.

- Aplicaciones de sistema

La estructura de datos para los sistemas operativos comprende muchas tablas, archivos y listas que tienen una estructura bien definida.

- Aplicaciones de CAD/CAM/CAE

Los sistemas de diseño/fabricación/ingeniería asistidos por computadoras requieren estructuras de datos sofisticadas para el almacenamiento, traducción y procesamiento de información.

- Aplicaciones del dominio de la ingeniería y de la ciencia.

El diseño orientado a la estructura de datos es mas difícil de aprender y mas complicado de aplicar que las técnicas orientadas al flujo de datos. Sin embargo, el diseño orientado a la estructura de datos ofrece un enfoque más rico y potencialmente, más poderoso para el diseño de software.

TEMA 7 Diseño Orientado al Objeto

En este tema trataremos el diseño orientado al objeto, año con año se logran avances que conducen a la construcción de computadoras mas veloces, mas compactas y baratas; sin embargo, el aspecto de software no se ha desarrollado en forma similar, mientras los costos de hardware han disminuido continuamente, los de software han hecho lo contrario. No es una tarea fácil la construcción de software y en varias ocasiones los proyectos de programación sobregiran los presupuestos de tiempo y dinero.

El problema del mantenimiento de software

Como ya hemos mencionado en temas anteriores la construcción de software ha sido la principal preocupación de los expertos por lo que se han conseguido avances significativos para la construcción de programas en forma sistemática y a bajo costo. Algunos de los resultados de estos esfuerzos son las técnicas que con el diseño estructurado y el desarrollo descendente (top-down), han sido utilizadas durante mucho tiempo por los programadores. Pero hay aun tres grandes deficiencias:

- los productos que resultan de emplear estas técnicas son poco flexibles

- los programadores que las usan tienden a concentrarse en el diseño e implementación inicial del sistema, sin tomar en cuenta su vida posterior.

- no alientan al programador a aprovechar el trabajo de proyectos anteriores.

Una desventaja de utilizar una metodología que se concentra en el diseño inicial del sistema se hace evidente si se considera que la vida útil de un producto de software puede ser cinco o seis veces mas grande que el lapso en que se desarrolla. Los gastos que se hacen durante el último período (gastos de mantenimiento) representan el 70% del costo total del sistema. Alrededor del 42% del costo total se tienen que hacer por cambios en las especificaciones del usuario o en los formatos de los datos. Es por ello que la extensibilidad (la facilidad con que se modifica un sistema para que realice nuevas funciones) debe ser uno de los objetivos primarios de la etapa de diseño.

En la fase de mantenimiento es muy importante que cualquier método de diseño tenga como objetivo principal producir sistemas que faciliten su propio mantenimiento. Para ello se recomienda una serie de actividades y heurísticas que pueden servir como guía durante el desarrollo del sistema.

Reutilización de código

Otro de los factores que no se toma en cuenta en los métodos de diseño tradicionales es la reutilización de código. Cualquier programador que desarrolla un sistema nuevo debe escribir una buena cantidad de código que ha escrito anteriormente una y otra vez. Las rutinas de búsqueda, ordenamiento, manejo de menús y despliegue de ventanas, se repiten continua-

mente en proyectos diferentes. Los métodos de desarrollo de software deben alentar al programador a utilizar el código escrito previamente por el mismo o por otros programadores.

Se han empleado tres tipos de reutilización:

- de personal: un proyecto nuevo se le asignan a programadores que tienen experiencia en el desarrollo de proyectos semejantes.

- de diseño: consiste en emplear el diseño de un sistema para desarrollar otro sistema similar

- de código fuente: se da cuando un programador utiliza parte de un programa escrito con anterioridad para crear un nuevo programa.

Sin embargo, estas tres formas de reutilización se han empleado en forma muy limitada, por lo que es necesario buscar técnicas mas generales.

Modularidad

En temas anteriores mencionamos una de las herramientas de diseño más poderosa para facilitar el desarrollo y mantenimiento de sistemas de software, la modularidad. En este tema trataremos algunas de las características de esta herramienta.

Como recordaran, la modularidad es dividir un sistema complejo en unidades mas pequeñas y manejables, cada una de esas unidades se encarga de manejar un aspecto local de todo el sistema, interactuando con otros módulos para cumplir con el objetivo global.

El concepto de módulo es aún mas sofisticado. Entre las propiedades que tiene están: la decomponibilidad, componibilidad, comprensibilidad, continuidad y protección.

a) Decomponibilidad: Un método modular debe permitir descomponer un problema de diseño en problemas más pequeños que pueden resolverse en forma independiente.

b) Componibilidad: una vez que se cuenta con un conjunto de módulos que realizan una función especifica, se debe alentar al programador a usar esos módulos para construir nuevos programas.

c) Comprensibilidad: El lector de un programa o librería debe ser capaz de entender el funcionamiento de cada módulo sin necesidad de consultar el texto de otros módulos.

d) Continuidad: un cambio pequeño en las especificaciones de un programa debe causar cambios en un sólo módulo ó en un conjunto pequeño de ellos.

e) Protección: un error de ejecución en el funcionamiento de un modulo no debe expandirse hacia los demás módulos.

Para alentar estas propiedades se utilizan las siguientes estrategias:

Primero: Un modulo debe corresponder con una unidad sintáctica del lenguaje.

Segundo: Los módulos deber tener pocas interfaces (medio de comunicación entre módulos) y estas debe ser pequeñas. Un número pequeño de interfaces aumenta la independencia de los módulos, haciendo mas fácil el proceso de componer nuevos sistemas a partir de módulos prefabricados, ayuda además a evitar que los errores de un módulo se propaguen por todo el sistema.

y **Tercero:** Cada módulo debe ocultar su implementación y algoritmos internos al resto del sistema. No se debe permitir que un módulo modifique los elementos internos de otros módulos.

Diseño orientado a objetos

El diseño orientado a objetos (DOO), es un método que tiene como objetivo establecer técnicas de desarrollo de software para producir sistemas modulares. Un sistema orientado a objetos se puede entender fácilmente, facilitando así su desarrollo, depuración y mantenimiento.

El DOO, se basa en un medida sencillas; un sistema de computadora es un modelo que representa un subconjunto del mundo real; la estructura de ese sistema se simplifica en gran medida si cada una de las entidades u objetos, del problema que se esta modelando corresponde directamente con un objeto que se pueda manipular internamente en el sistema.

El proceso de representar entidades reales con elementos internos en un sistema recibe el nombre de abstracción de datos.

La abstracción de datos no se concentra en la representación interna de un objeto (una cadena, un arreglo, un entero, etc.)

sino en sus atributos y en las operaciones que se pueden realizar sobre ese objeto. De esta forma, un tipo de dato abstracto se puede describir concentrándose en las operaciones que manipulan a los objetos de ese tipo, sin caer en los detalles de representación y manipulación de datos.

La creación de tipos abstractos permite al diseñador adaptar el diseño a sus necesidades especificas, se debe ser capaz de definir tipos de datos que se ajusten al problema que se esta resolviendo y que puedan ser manipulados después por el lenguaje. Concentrándose así en los objetos que manipula su sistema y las relaciones entre estos objetos, haciendo a un lado la representación y manipulación de datos, para comprender mejor el problema.

Conceptos

Bien, ahora para entender un poco mas acerca del diseño orientado a objetos (DOO), trataremos algunos conceptos:

CLASE:

Es un tipo abstracto del diseño orientado a objetos, una clase describe un conjunto de objetos semejantes. Dicha descripción se hace en dos partes:

- los datos que especifican las propiedades de los objetos de esa clase, llamados atributos.

- y las funciones que manipulan esos datos, llamados métodos.

Un objeto puede recibir mensajes de otros objetos; entonces, debe escoger uno de sus métodos y dar una respuesta basándose en los datos que representan su estado.

ELEMENTOS PRIVADOS Y PUBLICOS

Cada objeto cuenta con elementos privados, que solo pueden ser usados por objetos de su misma clase. Los elementos públicos representan la interface de un objeto con su medio ambiente, únicamente esos elementos pueden modificar a los datos privados, los cuales representan el estado del objeto.

Se proponen dos estrategias para la reutilización de código: la composición y la herencia.

La COMPOSICION permite definir una nueva clase de objetos mediante la unión de un conjunto de clases ya existentes.

La HERENCIA permite derivar una nueva clase basándose en otra clase mas general. Una clase derivada adquiere todas las propiedades y métodos de la clase de la que se derivo, llamada clase base.

Además de facilitar la reutilización de código, la herencia es un medio ideal para crear sistemas con alta extensibilidad. Otra ventaja de esta técnica es que permite manipular objetos de clases diferentes como si fueran de la misma clase, con lo cual es posible definir interfaces uniformes para diferentes tipos de objetos.

Lenguajes orientados a objetos

Las técnicas en las que se basa el diseño orientado a objetos como son ocultamiento de información, abstracción de datos, manejo de clase, eran conocidas y utilizadas por los ingenieros de software desde hace muchos años, sin embargo hay pocos lenguajes que brindan todas las facilidades para escribir programas orientados a objetos, algunos de los mas populares

son Simula67, Smalltalk, Eiffel, Ada o Modula. Estos lenguajes orientados a objetos no habían tenido hasta hace poco una aceptación amplia entre la comunidad de programadores.

TEMA 8. IMPLEMENTACION DEL SISTEMA

Introducción

Hasta el momento hemos aprendimos cómo se efectúa el análisis del sistema; es decir cómo elaborar un estudio de necesidades y requerimientos de nuestro sistema de información antes de desarrollarlo.

Posteriormente, conocimos las técnicas para diseñar, con base al análisis previo, nuestro sistema, orientándolo al medio en el que se desarrollará y aplicará.

Ahora sabremos cómo desarrollar nuestro sistema tomando en cuenta las especificaciones del diseño y el equipo de cómputo seleccionado.

En este tema conoceremos las formas de plasmar en un equipo de cómputo el diseño de un sistema. Para ello contamos con cuatro subtemas:

Introducción

Técnicas de codificación estructurada

Estándares de Implementación y

Documentación

Subtema 8.1 Introducción

Hasta ahora hemos hablado de cómo efectuar el análisis y el diseño de un sistema que van dirigidos hacia un objetivo final: la traducción de las especificaciones del diseño a una forma que puede ser "comprendida" por una computadora. Hemos llegado finalmente al paso de la codificación. El objetivo principal de la implementación del sistema es el escribir código fuente así como la documentación interna de modo que la concordancia del código con sus especificaciones sea fácil de verificar, así como facilitar la depuración, las pruebas y modificaciones al sistema. Para ello se debe elaborar un código claro utilizando 3 parámetros:

Técnicas de codificación estructurada

Estándares de implementación y

Documentación.

Cada uno de estos puntos lo tocaremos específicamente en los siguientes subtemas.

Subtema 8.2 Técnicas de codificación estructurada.

Las técnicas de codificación tienen como fin facilitar la traducción del diseño detallado de los módulos que componen un programa a un lenguaje de programación específico, manteniendo la estructura jerárquica definida en la fase de diseño.

La programación estructurada es la aplicación de métodos básicos de descomposición de problemas para establecer una estructura jerárquica fácilmente manejable a través del proceso de refinamiento progresivo. Es decir, cualquier módulo, aún el más complicado, se detalla utilizando las reglas básicas de la programación estructurada que son:

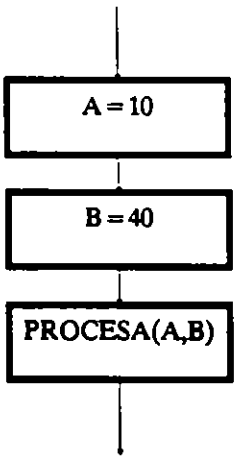
Secuencia

Decisión

Repetición

A continuación explicaré cada una de estas reglas.

La secuencia se define como una estructura de proceso que consiste en enunciar, una después de otra, una serie de instrucciones.



Gráficamente se representa como se muestra en la figura, en donde cada instrucción está encerrada en un cuadro y mantiene una jerarquía. Esto es, se van ejecutando las instrucciones una después de la otra.

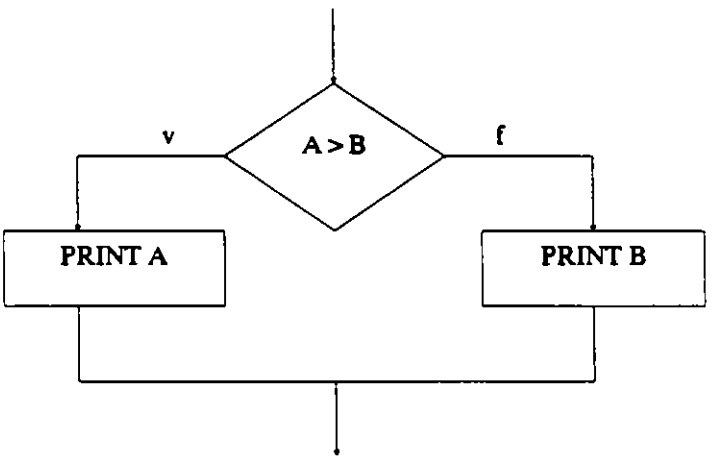
A = 10

B = 40

PROCESA(A,B)

Esta figura muestra un ejemplo en donde está parte de un programa que contiene instrucciones simples exclusivamente. Una proposición simple puede ser una asignación o un llamado a un procedimiento; una proposición compuesta es un if-then-else o un while-do. Todas estas proposiciones se pueden presentar como una secuencia.

La segunda regla básica, la decisión, es una estructura de proceso que permite especificar alternativas en la ejecución de instrucciones, dependiendo de una condición.



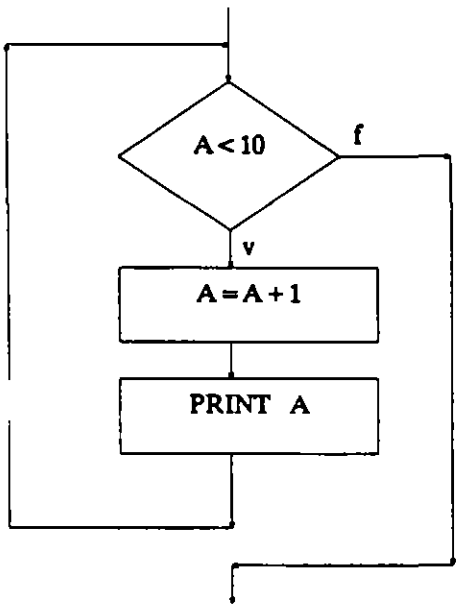
Gráficamente se representa como se ve en la figura, en donde la condición que determinará la alternativa a seleccionar está en un rombo y de éste se desprenden las alternativas a seguir.

Un ejemplo de codificación de una decisión se muestra a continuación. Las alternativas a realizar se presentan con cierta sangría para detectar con facilidad la dependencia de su ejecución con respecto a la condición. Comunmente se utiliza la proposición if-then-else para su construcción.

```
IF  A > B THEN
      PRINT A
ELSE
      PRINT B
ENDIF
```

Y la última regla básica, la repetición, es una estructura de proceso que permite la especificación de la ejecución iterativa de una serie de instrucciones.

En la figura se puede observar la representación gráfica de la repetición. La condición que determinará el número de iteraciones está encerrada en un rombo.



Al codificar una repetición, como se muestra en la figura, se deberán sangrar las instrucciones que se estarán iterando. Es común utilizar las proposiciones while-do o repeat-until para codificar la estructura de repetición.

```
WHILE  A < 10
     A = A + 1
     PRINT  A
END
```

Una vez que han quedado claras las tres reglas básicas de la codificación estructurada, podemos decir que la programación estructurada se basa en un teorema el cual establece lo siguiente:

Cualquier módulo de un programa con una entrada y una salida que tenga todas las proposiciones en algún camino de la entrada a la salida puede especificarse usando solamente secuenciación, decisión y repetición.

La propiedad de una entrada y una salida permite el anidamiento de construcciones dentro de otra en cualquier forma que se desee. Esto es, por ejemplo, si tenemos una proposición de decisión, una alternativa puede ser otra proposición de decisión que a su vez una de sus alternativas sea otra proposición de decisión. La codificación correspondiente a este ejemplo se muestra en la figura.

```
WHILE  A>B
       A=A-1
       WHILE  B>0
              B=A/B
              WHILE B>1
                     PRINT B
              END
       END
END
```

El proceso de la codificación, que es la comunicación entre los humanos y las computadoras mediante un lenguaje de programación, es una actividad humana y es por ello que las características del lenguaje afectan directamente a la calidad de la comunicación, tienen un impacto importante sobre el éxito de un proyecto de desarrollo de software y pueden influenciar la calidad del diseño (ya que prácticamente el diseño detallado se dirige hacia un lenguaje de programación específico).

La elección de un lenguaje de programación para un proyecto específico debe tener en cuenta sus características y ciertos criterios.

El problema asociado con la elección del lenguaje puede desaparecer si solo se dispone de un lenguaje o si el cliente demanda uno en particular.

Los criterios que se aplican durante la evaluación de los lenguajes disponibles están:

1) area de aplicación general

2) complejidad algorítmica y computacional

3) entorno en el que se ejecutará el software

4) consideraciones de rendimiento

5) complejidad de las estructuras de datos

6) conocimiento de la plantilla de desarrollo de software y

7) disponibilidad de un buen compilador o compilador cruzado.

Actualmente C es el lenguaje elegido comunmente en el desarrollo de software de sistemas.

Y volviendo con las técnicas de codificación estructurada podemos decir que en un nivel fundamental, todos los lenguajes modernos permiten al programador representar secuencias, decisiones y repeticiones -que son las construcciones lógicas de la programación estructurada. La mayoría de los lenguajes de programación modernos proporcionan un sintáxis para la especificación directa del if then else, del do while y del repeat until. Otros lenguajes, como Lisp y APL, requieren que el programador emule esas construcciones dentro de los límites de sintaxis del lenguaje.

Con frecuencia se menciona que el apegarse estrictamente a las técnicas de codificación estructurada existe la necesidad de emplear variables auxiliares, segmentos de código repetido y un excesivo llamado a subprogramas, dando como resultado una utilización ineficiente del espacio de memoria y del tiempo de ejecución. Es por ello que hablaré de la eficiencia de código.

La eficiencia del código fuente está directamente unida a la eficiencia de los algoritmos definidos durante el diseño detallado. Sin embargo, el estilo de codificación puede afectar a la velocidad de ejecución y a los requerimientos de memoria. Las siguientes directrices se deben seguir siempre en el proceso de traducción del diseño detallado al código:

- Simplificar las expresiones aritméticas y lógicas antes de convertirlas en código.

- Evaluar cuidadosamente las iteraciones anidadas para determinar si se pueden sacar fuera de ellos algunas sentencias o expresiones.

- Cuando sea posible, evitar el uso de arreglos multidimensionales.

- Cuando sea posible, evitar el uso de apuntadores y listas complejas.

- Usar rápidas operaciones aritméticas.

- Usar cuando sea posible aritmética entera y expresiones Booleanas o lógicas.

Por lo anterior, podemos decir que las técnicas de codificación estructurada no son en sí las que propician la ineficiencia sino los algoritmos y las estructuras de datos los que afectan la velocidad de ejecución y el espacio en memoria respectivamente.

Subtema 8.3 Estándares de Implementación.

A continuación veremos qué son y para qué nos sirven los estándares de implementación, dentro de la implementación de un sistema.

Los estándares de codificación son especificaciones para un estilo de codificación preferido. Dada una situacion de elegir los caminos para lograr un efecto, se especifica un camino preferido. Los estándares de codificación a menudo son vistos por los programadores como mecanismos para restringir y devaluar las habilidades para resolver problemas creativos de los programadores. La creatividad siempre ocurre dentro de un marco de trabajo básico de estándares. Los artistas siguen los principios básicos de estructura y composición, los poetas se apegan al ritmo y a la métrica del lenguaje. Sin un marco de trabajo de estándares que guíen y canalicen una actividad, la creatividad se vuelve un caos sin sentido.

Así, es **deseable** que todos los programadores de un proyecto adopten un estilo de codificación similar, de modo que se produzca un código de calidad uniforme.

Algunos de los estándares que se pueden aplicar en todas las situaciones son:

- El uso de la instrucción GOTO debe evitarse en circunstancias normales. Esto es, utilizarse únicamente para lograr el formato de las construcciones de la codificación estructurada en los lenguajes primitivos. La proposición GOTO es valiosa para las salidas múltiples y prematuras de los ciclos, y para transferir el control al código manipulador de errores. Como pueden observar en su pantalla tenemos primeramente un código en el que se emplea la instrucción GOTO sin necesidad y posteriormente cómo debe ser.

- Introdúzcanse tipos de datos definidos por el usuario. Muchos lenguajes de programación proporcionan tipos de datos definidos por el usuario. El uso de distintos tipos de datos hace posible distinguir las entidades propias del problema. Por ejemplo si definimos un tipo de datos llamado color y declaramos los colores: amarillo, rojo, azul, verde y anaranjado; podemos saber que únicamente, de toda la gama de colores que existen, se van a utilizar esos.

- Examínense cuidadosamente las rutinas que tengan menos de 5 o más de 25 proposiciones. Un subprograma que tiene más de 25 proposiciones ejecutables probablemente contiene una o más subfunciones bien definidas que pueden agruparse como subprogramas distintos e invocarse cuando se necesiten. Se dice también que 30 proposiciones ejecutables es el límite superior de compresión para una sola lectura de un subprograma escrito en un lenguaje de programación. Un subprograma que tiene menos de cinco proposiciones normalmente es muy pequeño para proporcionar una función bien definida. Existen excepciones obvias a estos principios generales: por ejemplo rutinas para obtener el valor absoluto o el módulo de un número y que tiene menos de 5 líneas pueden ser muy útiles.

- Utilíncese sangrías, paréntesis, espacios y líneas en blanco, así como márgenes alrededor de los bloques de comentarios para aumentar la legibilidad. Si el formato del texto fuente lo hacen varios programadores, todos ellos deben acordar y apegarse a condiciones estándar.

- Proporciónense prólogos estándares de documentación para cada subprograma. Un prólogo de documentación contiene la información acerca de un subprograma o una unidad de compilación que no resulta obvia al leer el texto fuente del subprograma o unidad de compilación.

- Aíslense las dependencias de la máquina en unas cuantas rutinas. Si se necesita cambiar la naturaleza de la representación de datos, o si el programa se cambia a una máquina diferente, se localizan los detalles de la dependencia de la máquina y pueden modificarse en consecuencia.

- La profundidad de anidamiento de la construcciones de un programa debe ser de cinco o menos. Si el anidamiento se vuelve demasiado profundo, se vuelve difícil determinar las condiciones bajo las cuales se va a ejecutar la proposición más anidada.

La claridad de un programa depende en gran medida, del estilo del programador. Si bien los principios de un buen estilo de programación no pueden ser expresados como reglas mecánicas de la buena programación, la experiencia nos indica que los estándares pueden servir como "guías de estilo".

Subtema 8.4 Documentación.

La programación por computadora incluye el código fuente de un sistema y todos los documentos de apoyo generados durante el análisis, diseño, implementación, pruebas y mantenimientodel sistema.

Las especificaciones de requisitos, documentos de diseño, planes de prueba, manuales de usuario y los reportes de mantenimiento son ejemplos de documentos de apoyo. Estos documentos son los productos que resultan del desarrollo y mantenimiento sistemático de la programación.

Un enfoque sistemático al desarrollo de la programación garantiza que los documentos de apoyo se desarrollen de una manera ordenada y que esos documentos se encuentren disponibles cuando se necesiten. Estos documentos deben desarrollarse como un producto natural paralelo al proceso de desarrollo. La calidad, cantidad, duración y utilidad de los documentos de apoyo son las principales medidas de la salud y la bondad de un proyecto de programación.

Los documentos sirven para explicar y/o aclarar lo que sucede en el flujo de control de un programa. Existen dos tipos de Documentación:

- Documentación interna

- Comentarios explicativos del cuerpo del módulo.

La documentación interna consiste en un prólogo estándar para cada unidad de programa, los aspectos autodocumentados del código fuente y los comentarios internos intercalados en la porción ejecutable del código.

La mejor documentación de un programa de computadora la constituye la claridad de su estructura; además la documentación más confiable de un programa es el código mismo. Por lo tanto se recomienda la selección de lenguajes que estimulen una programación legible y estructurada, evitando la necesidad de instrucciones de flujo arbitrariamente complicadas.

El formato característico de los prólogos de subprogramas es:

Nombre del autor:

Fecha de terminación:

Función realizada:

Rutinas que lo invocan:

Rutinas que invoca:

Autor/fecha/modificación:

Resumen

La implementación de un sistema tiene por objetivo obtener un código fuente claro, que facilite la depuración, prueba y modificación y que estas actividades consuman una gran porción de la mayor parte de los presupuestos en la programación.

En este tema se analizaron las técnicas de la codificación estructurada, el estilo de codificación, estándares y principios generales y la documentación interna y de apoyo del código fuente.

TEMA 9. Verificación y validación del Sistema

En el tema anterior conocimos las técnicas y consideraciones para implantar nuestro sistema de información en un equipo de cómputo elegido; sin embargo es muy importante efectuar verificaciones o pruebas en su implementación para obtener un sistema confiable y eficiente. El desarrollo de sistemas de software envuelve una serie de actividades de producción en las que las posibilidades de que aparezca la falla humana son enormes. Los errores pueden darse desde el primer momento del proceso en el que los objetivos se pueden especificar de forma errónea o imperfecta, así como los errores que aparecen en los posteriores pasos de diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad.

Los objetivos de las actividades de verificación y validación son el valorar y mejorar la calidad de los productos de trabajo generado durante el desarrollo del diseño y la implementación.

La verificación se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica. La validación se refiere a un conjunto diferente de actividades que aseguran que el software construído se ajusta a los requerimientos del cliente. Podemos establecer como

Verificación al hacer la pregunta: ¿Estamos construyendo correctamente el producto? es decir, cómo estamos construyendo el producto, ¿de manera adecuada?. Mientras que para establecer a la validación podemos hacer la pregunta: ¿Estamos construyendo el producto correcto? es decir, el producto que se está elaborando es el que cubre todos los requerimientos.

La calidad de los productos de trabajo generados durante el análisis y el diseño se pueden estimar y mejorar utilizando procedimientos sistemáticos de control de calidad, mediante recorridos e inspecciones y por medio de verificaciones automatizadas para supervisar que sea consistente y que esté completo. La técnica para estimar y mejorar la calidad del código fuente incluye los procedimientos sistemáticos de la prueba de unidad, la prueba de integración, la prueba de validación y la prueba del sistema.

Subtema 9.1 Pruebas del Sistema.

Ahora detallaré cada una de las pruebas que se mencionaron anteriormente.

Iniciaremos con la Prueba de Unidad. La prueba de unidad centra el proceso de verificación en la menor unidad de diseño de software. Las pruebas de unidad comprenden el conjunto de pruebas efectuadas por un programador individual, antes de la integración de la unidad en un sistema más grande. Una unidad de programa suele ser lo suficientemente pequeña como para que el programador que la desarrolló pueda probarla con minuciosidad. Las pruebas que se dán como parte de la prueba de unidad son:

Prueba de interfaz

Prueba de las estructuras de datos

Pruebas de condiciones de límite

Pruebas de caminos independientes

Manejos de errores

Veamos ahora **Pruebas de integración.** Las pruebas de integración es una técnica sistemática para construir la estructura del sistema mientras que al mismo tiempo se llevan acabo pruebas para detectar errores asociados con la interacción entre sus módulos. El objetivo es tomar los módulos probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Existen dos estrategias de integración:

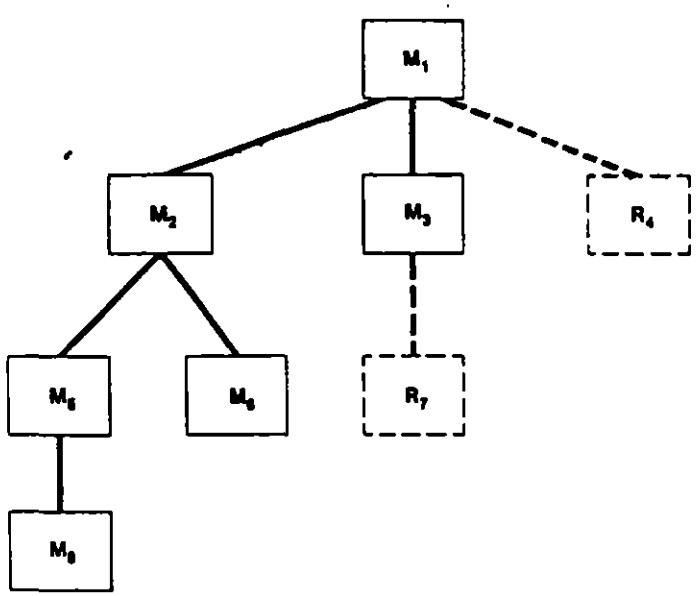
-Integración descendente

-Integración ascendente

Veremos cada una de ellas a continuación; comenzaremos primeramente con la Integración descendente.

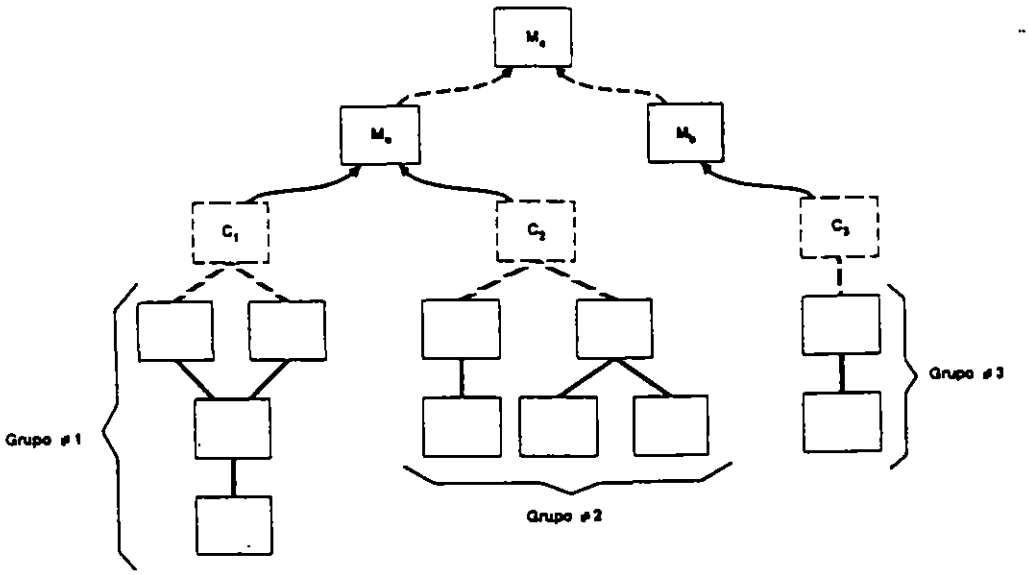
En la integración descendente se va haciendo una aproximación incremental a la construcción de la estructura de programas. Es decir, se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando con el módulo de control principal (llamado comúnmente programa principal). Los módulos subordinados al módulo de control principal se van incorporando en la estructura, ya sea de la forma primero-en-profundidad o de la forma primero-en-anchura

En la integración primero-en-profundidad se integran todos los módulos de un camino de control principal de la estructura. Para seleccionar el camino principal se puede hacer de una manera arbitraria y dependerá de las características específicas de la aplicación.



4o. Se eliminan los conductores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

Veámoslo a manera ilustrativa con el siguiente ejemplo: Siguiendo el esquema que se muestra en la figura, se combinan los módulos para formar los grupos 1, 2, y 3. Cada uno de los grupos se somete a prueba mediante un conductor que se muestra como un bloque punteado. Los módulos de los grupos 1 y 2 son subordinados de Ma. Se eliminan los conductores C1 y C2 y se conectan los grupos directamente a Ma. De forma similar se elimina el conductor C3 del grupo 3 antes de integrarlo con el módulo Mb. Finalmente, tanto Ma como Mb se integran en el Módulo Mc, y así sucesivamente.



A medida que la integración progresa hacia arriba, disminuye la necesidad de conductores de prueba separados. De hecho, si los niveles superiores del programa se integran de forma descendente, se puede reducir sustancialmente el número de conductores y se simplifica enormemente la integración de grupos.

Revisemos ahora la **Prueba de Validación.**

Tras la culminación de la prueba de integración, el software está completamente ensamblado como un paquete; se han encontrado y corregido los errores de interfaces, y debe comenzar una serie final de pruebas de software **la prueba de validación** y ésta se logra cuando el software funciona de acuerdo a las espectativas razonables del cliente. Para la prueba de validación tenemos que considerar 3 puntos.

Criterios para la prueba de validación.

Repaso de la configuración.

Pruebas alfa beta.

Por último pasemos a describir la Prueba del Sistema.

Una vez que el software es incorporado a otros elementos del sistema y se realiza una serie de pruebas de integración del sistema y de validación, se realizan las pruebas generales del Sistema.

Los pasos dados durante el diseño del software y durante la prueba pueden mejorar enormemente la probabilidad de éxito en la integración del software en el sistema total.

La prueba del sistema realmente está constituída por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en la computadora. Los tipos de prueba del sistema son:

Prueba de recuperación

Prueba de seguridad

Prueba de resistencia

Prueba de rendimiento

Bien, en este tema conocimos las pruebas que hay que aplicar ha nuestro sistema tanto en el momento de su desarrollo como cuando se ha integrado totalmente con la finalidad de obtener un sistema de alta calidad.

LECTURA: PRUEBAS DE UNIDAD.

En las pruebas de unidad se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad del programa que está siendo probada. Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento. Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y finalmente, se prueban todos los caminos de manejo de errores.

Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del módulo. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. Myers, en su libro sobre prueba de software, propone una lista de comprobaciones para la prueba de interfaces:

1. ¿Es igual el número de parámetros de entrada al número de argumentos?
2. ¿Coinciden los atributos de los parámetros y los argumentos?
3. ¿Coinciden los sistemas de unidades de los parámetros y de los argumentos?
4. ¿Es igual el número de argumentos transmitidos a los módulos de llamada que el número de parámetros?
5. ¿Son iguales los atributos de los argumentos transmitidos a los módulos de llamada y los atributos de los parámetros?
6. ¿Son iguales los sistemas de unidades de los argumentos transmitidos a los módulos de llamada y de los parámetros?
7. ¿Son correctos el número, los atributos y el orden de los argumentos de las funciones incorporadas?
8. ¿Existen referencias a parámetros que no estén asociados con el punto de entrada actual?
9. ¿Entran sólo argumentos alterados?
10. ¿Son consistentes las definiciones de variables globales entre los módulos?
11. ¿Se pasan las restricciones como argumentos?

Cuando un módulo lleve a cabo E/S externa, se deben llevar a cabo pruebas de interfaz adicionales. De nuevo, de Myers:

1. ¿Son correctos los atributos de los archivos?
2. ¿Son correctas las sentencias de apertura?
3. ¿Cuadran las especificaciones de formato con las sentencias de E/S?
4. ¿Cuadra el tamaño del buffer con el tamaño del registro?
5. ¿Se abren los archivos antes de usarlos?
6. ¿Se tienen en cuenta las condiciones de fin de archivo?
7. ¿Se manejan los errores de E/S?
8. ¿Hay algún error textual en la información de salida?

Las estructuras de datos locales de cada módulo son una fuente potencial de errores. Se deben diseñar casos de prueba para descubrir errores de las siguientes categorías.

1. tipificación impropia o inconsistente
2. iniciación o valores implícitos erróneos

3. nombres de variables incorrectos (mal escritos o truncados)
4. tipos de datos inconsistentes
5. excepciones de desbordamiento por arriba o por abajo, o de direccionamiento

Además de las estructuras de datos locales, durante la prueba de unidad se debe comptobar (en la medida de lo posible) el impacto de los datos globales sobre el módulo (p.ej., el COMMON de FORTRAN).

Durante la prueba de unidad, la comprobación selectiva de los caminos de ejecución es una tarea escencial. Se deben diseñar casos de prueba para detectar errores debido a cálculos incorrectos, comparaciones incorrectas o flujo de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos.

Entre los errores más comunes en los cálculos estan: 1) precedencia incorrecta o mal interpretada; 2) operaciones de modo mixto; 3) inicializaciones incorrectas; 4) falta de precisión; 5) incorrecta representación simbólica de una expresión. Las comparaciones y el flujo de control están fuertemente emparejadas (p. ej., el flujo de control cambia frecuentemente tras una comparación). Los casos de prueba deben descubrir errores como: 1) comparaciones entre tipos de datos distintos; 2) operadores lógicos o precedencia incorrectos; 3) igualdad operada cuando los errores de precisión la hacen poco probable; 4) variables o comparadores incorrectos; 5) terminación de bucles inapropiada o inexistente; 6) fallo de salida cuando se encuentra una iteración divergente, y 7) variables de bucles modificadas de forma inapropiada.

Un buen diseño dicta que las condiciones de error sean previstas de antemano y que se dispongan unos caminos de manejo de errores que dirijan o terminen de una forma limpia el procesamiento cuando se dé un error.

Entre los errores potenciales que se deben comprobar cuando se evalúa la manipulación de errores están:

1. Descripción ininteligible del error
2. El error señalado no se corresponde con el error encontrado.
3. La condición de error hace que intervenga el sistema antes que el mecanismo de manejo de errores
4. El procesamiento de la condición excepcional es incorrecto.
5. La descripción del error no proporciona suficiente información para ayudar en la localización de la causa del error.

La prueba de límites es la última (y probablemente la más importante) tarea del paso de prueba de unidad. El software falla en sus condiciones límites. O sea, a menudo aparece un error cuando se procesa el elemento n-simo de un arreglo n-dimensional, cuando se hace la i-ésima repetición de un bucle de i pasos o cuando se encuentra los valores máximo o mínimo permitidos. Los casos de prueba que ejerciten las estructuras de datos, el flujo de control y los valores de los datos por debajo, en y por encima de los máximos y los mínimos son muy apropiados para descubrir estos errores.

LECTURA: PRUEBAS DE VALIDACION.

La validación del software se consigue mediante una serie de pruebas que demuestran la conformidad con los requerimientos. Un plan de prueba traza las pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos que serán usados para demostrar la conformidad con los requerimientos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requerimientos funcionales, que se alcanzan todos los requerimientos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requerimientos (p. ej., portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento).

Una vez que se procede con cada caso de prueba de validación, puede darse una de dos condiciones: 1) Las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptables, o 2) Se descubre una desviación de las especificaciones y se crea una lista de deficiencias. Las desviaciones o errores descubiertos en esta fase del proyecto raramente se pueden corregir antes de terminar el plan. A menudo es necesario negociar con el cliente un método para resolver las deficiencias.

Repaso de la configuración

Un elemento importante del proceso de validación es el repaso de la configuración. El repaso intenta asegurar que todos los elementos de la configuración del software se han desarrollado de forma adecuada, están catalogados y tienen el suficiente detalle para facilitar la fase de mantenimiento dentro del ciclo de vida del software.

Pruebas alfa y beta

Es virtualmente imposible que un encargado del desarrollo del software pueda prever cómo un cliente usará realmente un programa. Se pueden interpretar mal las instrucciones de uso, se pueden usar regularmente extrañas combinaciones de datos y una salida que pueden estar clara para el que realiza la prueba puede resultar ininteligible para un usuario normal.

Cuando se construye software a medida para un cliente, se lleva a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requerimientos. Llevado a cabo por el usuario final en lugar del equipo de desarrollo, una prueba de aceptación puede ir desde un informal "paso de prueba" hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, la prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema.

Si el software se desarrolla como un producto a ser usado por muchos clientes, no es práctico realizar pruebas de aceptación formales para cada uno de ellos. La mayoría de los constructores de productos de software llevan a cabo un proceso denominado prueba alfa y beta para descrubrir errores que parezcan que sólo el usuario final puede descubrir.

La prueba alfa es conducida por un cliente en el lugar de

desarrollo. Se usa el software de forma natural, con el encargado del desarrollo "mirando por encima del hombro" del usuario y registrando errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.

La prueba beta se lleva a cabo en uno o más lugares de clientes por los usuarios finales del software. A diferencia de la prueba alfa, el encargado del desarrollo normalmente no está presente. Así, la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el equipo de desarrollo. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al equipo de desarrollo. Como resultado de los problemas anotados durante la prueba beta, el equipo de desarrollo del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la base de clientes.

LECTURA: PRUEBA DEL SISTEMA.

Prueba de recuperación

Muchos sistemas basados en computadora deben recuperarse de los fallos y resumir el procesamiento en un tiempo previamente especificado. En algunos casos un sistema debe ser tolerante a fallos de procesamiento no deben hacer que cese el funcionamiento de todo el sistema. En otros casos, se debe corregir un fallo del sistema en un determinado período de tiempo a riesgo de que se produzca un serio daño económico.

La prueba de recuperación es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. Si la recuperación es automática (llevada a cabo por el propio sistema) hay que evaluar la corrección de la reinicialización, de los mecanismos de recuperación del estado del sistema, de la recuperación de datos y del arranque. Si la recuperación requiere la intervención humana, hay que evaluar los tiempos medios de reparación para determinar si están dentro de unos límites aceptables.

Prueba de seguridad

Cualquier sistema basado en computadora que maneje información sensible o lleve a cabo acciones que puedan impropiamente perjudicar (o beneficiar) a los individuos es objeto de penetraciones impropias o ilegales. La penetración incluye un amplio rango de actividades: "destripadores" que intentan penetrar sistemas como deporte, empleados disgustados que intentan penetrar por venganza e individuos deshonestos que intentan penetrar por obtener ganancias personales ilícitas.

La prueba de seguridad intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de la penetración impropia. Citando a Beizer [BEI84]: "Por supuesto, la seguridad del sistema debe ser probada en su invulnerabilidad frente a un ataque frontal pero también debe ser probada en su invulnerabilidad frente a ataques por los flancos o por la retaguardia".

Durante la prueba de seguridad, el encargado de la prueba juega el papel de un individuo que desea penetrar en el sistema. ¡Todo vale! Debe intentar hacerse con las claves de acceso por cualquier medio externo del oficio; debe atacar al sistema con software a medida, diseñado para romper cualquier defensa que haya sido construida; debe bloquear el sistema, negando así el servicio a otras personas; debe producir a propósito errores del sistema, intentando penetrar durante la recuperación; o debe curiosear en los datos públicos, intentando encontrar la clave de acceso al sistema.

Con el suficiente tiempo y los suficientes recursos, una buena prueba de seguridad terminará por penetrar en el sistema. El papel del diseñador del sistema es hacer que el coste de penetración sea mayor que el valor de la información obtenida mediante la penetración.

Prueba de resistencia

Durante los anteriores pasos de prueba, la técnica de la caja

blanca y de la caja negra daban como resultado la evaluación del funcionamiento y del rendimiento normales del programa. Las pruebas de resistencia están diseñadas para enfrentar a los programas con situaciones anormales. En esencia, el sujeto que realiza la prueba de resistencia se pregunta: "¿A qué potencia puedo ponerlo a funcionar antes de que falle?

La prueba de resistencia ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo: 1) diseñar pruebas especiales que generan diez interrupciones por segundo, mientras que una o dos son las normales; 2) incrementar las frecuencias de datos de entrada en un orden de magnitud con el fin de comprobar cómo responden las funciones de entrada; 3) ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos; 4) diseñar casos de prueba que puedan dar problemas con el esquema de gestión de memoria virtual; 5) diseñar casos de prueba que produzcan excesivas búsquedas de datos residentes en disco. Esencialmente, el encargado de la prueba intenta tirar abajo el programa.

Una variación de la prueba de resistencia es una técnica denominada prueba de sensibilidad. En algunas situaciones (la más común se da con algoritmos matemáticos) un rango muy pequeño de datos dentro de los límites de una entrada válida para un programa puede producir un procesamiento extremo e incluso erróneo o una profunda degradación del rendimiento. Esta situación es análoga a la existencia de una signularidad en una función matemática. La prueba de sensibilidad intenta descubrir combinaciones de datos dentro de una clase de entrada válida que pueda producir inestabilidad o procesamiento incorrecto.

Prueba de rendimiento

Para sistemas de tiempo real o sistemas empotrados, el software que proporciona las funciones requeridas pero que no se ajusta a los requerimientos de rendimiento es inaceptable. La prueba de rendimiento está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos del proceso de prueba. Incluso al nivel de unidad, se debe asegurar el rendimiento de los módulos individuales a medida que se llevan a cabo las pruebas de la caja blanca. Sin embargo, hasta que no están completamente integrados todos los elementos del sistema no se puede asegurar realmente el rendiminento del sistema.

Las pruebas de rendimiento a menudo van emparejadas con las pruebas de resistencia y a menudo requieren instrumentación tanto de software como de hardware. O sea, a menudo es necesario medir la utilización de recursos (p. ej., ciclos de procesador) de una forma exacta. La instrumentación externa puede monitorizar los intervalos de ejecución, los sucesos ocurridos (p. ej., interrupciones) y muestras de los estados de la máquina en un funcionamiento normal. Instrumentando un sistema, el encargado de la prueba puede descubrir situaciones que lleven a degradaciones y posibles fallos del sistema.

LECTURA: MANTENIMIENTO DEL SISTEMA.

Mantenimiento estructurado frente al no estructurado

Si sólo se dispone del código fuente como elemento de configuración, la actividad de mantenimiento comienza con una dolorosa evaluación del código, a menudo complicada por la pobre documentación interna. Las sutiles caractrísticas, tales como la estructura del programa, las estructuras de datos globales, las interfaces del sistema, el rendimiento y/o las limitaciones del diseño, son dificiles de descubrir y frecuentemente mal interpretadas. Es difícil asegurar cuáles son las ramificaciones de cambios llevados últimamente en el código. Es imposible llevar a cabo pruebas de regresión (repetir prueba anterior para asegurar que las modificaciones no han introducido fallos en el software previamente operativo), ya que no existe ningún registro de pruebas. Lo que hacemos es un mantenimiento no estructurado y pagamos el precio (en esfuerzo desperdiciado y en frustraciones personales) que se adjunta a todo software que no ha sido desarrollado mediante una metodología bien definida.

Si existe una completa configuración del software, la tarea de mantenimiento comienza con una evaluación de la documentación del sideño. Se determinan las importantes caractrísticas estructurales, de rendimiento y de interfaz del software. Se estudia el impacto de las correciones o modificaciones requeridas y se traza un plan de actuación. Se modifica el diseño (usando técnicas idénticas a las discutidas en capítulos anteriores) y se revisa. Se desarrolla nuevo código fuente, se realizan pruebas de regresión mediante la información contenida en la Especificación de Prueba y se vuelve a lanzar el software.

Esta secuencia de sucesos constituye el mantenimiento estructurado y aparece como resultado de una anterior aplicación de una metodología de ingeniería del software. Aunque la existencia de una configuración del software no garantiza un mantenimiento libre de problemas, se reduce la cantidad de esfuerzo requerido y se mejora la calidad general del cambio o de la corrección .

Costos de mantenimiento

El costo del mantenimiento de software ha crecido rápidamente durante los últimos veinte años. Los porcentajes de factura total del software que se gastan en el mantenimiento de software existente son: para los años 70's de 35-40%, para los años 80's de 40-60% y se estima que para los años 90's será de 70-80%. Aunque los promedios industriales son difíciles de asegurar y están abiertos a muchas interpretaciones, una organización de desarrollo de software típica gasta entre el 40 y el 70 por ciento de todo su dinero en el mantenimiento correctivo, adaptivo, perfectivo y preventivo.

Los costos en dólares del mantenimiento son los más obvios. Sin embargo, otros cotos, menos tangibles, pueden en último lugar ser la causa de muchas preocupaciones. Citando a Daniel McCracken [MCC80]:

> Los retrasos en las nuevas ampliaciones y en los grandes cambios medidos en años se van alargando. Como industria, ni siquiera podemos hacer frente -digamos sólo poner al día- a lo que nuestros usuarios quieren que hagamos.

McCracken alude a la organización en la barrera del mantenimiento. Un costo intangible del mantenimiento del software viene dado por una oportunidad de desarrollo que se propone o se pierde debido a que los recursos disponibles deben estar dedicados a las tareas de mantenimiento. Otros costos intangibles incluyen:

- insatisfacción del cliente cuando una petición de reparación o de modificación aparentemente legítima no se puede antender en un tiempo razonable
- disminución de la calidad global del software debido a los errores latentes que introducen los cambios en el software mantenido
- trastornos en otros esfuerzos de desarrollo al tener que "poner" a trabajar a la plantilla en tareas de mantenimiento.

El costo final del mantenimiento de software es una dramática reducción de la productividad (medida en LDC por persona-mes o en punto de función por persona mes) que se encuentra cuando se inicia el mantenimiento de viejos programas. Se ha informado de reducciones de la productividad de 40 a 1 [BOE79]. O sea, un esfuerzo de desarrollo que haya resultado en un costo de $25 por línea de código desarrollada, costaría $1.000 por cada línea de código que sea mantenida.

El esfuerzo gastado en el mantenimiento se puede dividir en actividades productivas (p.el., análisis y evaluación, modificación del diseño, codificación) y actividades "menos productivas" (p. eje., tratar de comprender lo que es el código; tratar de interpretar las estructuras de datos, las características de la interfaz, los límites de rendimiento). La siguiente expresión [BEL72] proporciona un modelo del esfuerzo de mantenimiento:

$$M = p + K \exp(c-f)$$

donde M = esfuerzo total gastado en el mantenimiento
p = esfuerzo productivo (descrito arriba)
k = una constante empírica
c = una medidad de la complejidoad que se puede atribuir a la falta de un buen diseño y de documentación
f = una medida del grado de familiaridad con el software.

Este modelo indica que el esfuerzo (y costo) puede aumentar exponencialmente si se usa una pobre aproximación de desarrollo de software (o sea, una falta de ingeniería del software) y si la persona o grupo que usan la aproximación no está disponible para llevar a cabo el mantenimiento.

Problemas

La mayoría de los problemas asociados con el mantenimiento de software se debe a las deficiencias de la forma en que el software ha sido definido y desarrollado. Aquí aparece el clásico sindrome del pague ahora o pague después" La falta de control y disciplina en las dos primeras fases del proceso de ingeniería del software

casi siempre se traduce en problemas para la última fase.

Entre los muchos problemas clásicos asociados con el mantenimiento de software se encuentran los siguientes:

- A menudo es excepcionalmente dificil comprender un programa "ajeno". A medida que existen menos elementos de configuración del software, mayor es la dificultad. Si sólo existe el código indocumentado, es de esperar que aparezcan serios problemas.
- Ese personaje "ajeno" a menudo no se encuentra alrededor para poder explicarse. La movilidad entre los profesionales del software es actualmente muy grande. No podemos esperar una explicación personal del software por el que lo ha desarrollado una vez que se requiere el mantenimiento.
- No existe una documentación apropiada o está mal preparada. Un primer paso es el reconocimiento de que el software debe ser documentado, pero para que la documentación sea de algún valor debe ser comprensible y consistente con el código fuente.

- La mayoría del software no ha sido diseñado previendo el cambio. A menos que el método de diseño prevea el cambio mediante conceptos tales como independencia funcional o clases de objetos, las modificaciones del software serán dificiles y propensas a errores.

- El mantenimiento no se ve como un trabajo atractivo. Muchas de las causas de esto vienen dadas por el alto nivel de frustración asociado con el trabajo de mantenimiento.

Todos los problemas que se acaban de describir se pueden, en parte, atribuir al gran número de programas actualmente existentes que ha sido desarrollados sin tener en cuenta la ingeniería del software. Tampoco se debe ver como una panacea el uso de una metodología disciplinada. Sin embargo, la ingeniería del software por lo menos proporciona soluciones parciales a cada problema asociado con el mantenimiento.

Como consecuencia de los problemas asociados con el mantenimiento de software, surgen varias cuestiones técnicas y de organización. ¿Es posible desarrollar software que esté bien diseñado y sea fácil de mantener? ¿Podemos mantener la integridad del software cuando tiene que ser modificado? ¿Existen enfoques técnicos y organizativos que se puedan aplicar con éxito al mantenimiento de software? En las secciones que vienen a continuación se discuten estos y otros puntos.

ESTRATEGIAS DE DESARROLLO E IMPLEMENTACION DE SISTEMAS

1.1 ESTRATEGIAS DE DESARROLLO E IMPLEMENTACION DE SISTEMAS

La ingeniería de *software* es un conjunto de actividades cuyo objetivo es el desarrollo de *software* más confiable, menos costoso, y que funcione eficientemente sobre máquinas reales. Abarca un conjunto de tres elementos:

- métodos
- herramientas
- procedimientos

Los métodos abarcan la planificación y estimación de proyectos, análisis de los requerimientos del sistema y del *software*, diseño de las estructuras de datos, arquitectura de programas y procedimientos algorítmicos, prueba y mantenimiento.

Las herramientas de la ingeniería de *software* suministran un soporte automático o semiautomático para los métodos. Hoy en día, existen herramientas para soportar cada uno de los diversos métodos.

Los procedimientos unen a los métodos y herramientas y facilitan un desarrollo racional y oportuno del *software* de computadora. Definen la secuencia en la que se aplican los métodos, los controles que ayudan a asegurar la calidad y coordinación de los cambios en el *software*.

Las herramientas serán analizadas a detalle en el capítulo siguiente, por otra parte dado que los procedimientos vienen inherentes tanto a la metodología como a las herramientas electas, estos no serán desarrollados con tanta profundidad.

METODOS ESTRUCTURADOS

Los métodos estructurados consisten de un conjunto de reglas, tareas, y técnicas, para ayudar en el desarrollo de sistemas en uno o más de sus estados (estrategia, análisis, diseño, etc).

Los métodos estructurados empezaron a ser utilizados en círculos académicos a finales de los 60's, y alcanzaron popularidad en el ámbito comercial a mediados de los 70's. Estos métodos ofrecieron una solución a los entonces existentes estándares de desarrollo no estructurados y carentes de diseño.

Dentro de los métodos estructurados se encontraban las gráficas estructuradas de Yourdon, diagramas de Warnier/Orr, diagramas de Jackson, la técnica estructurada de Chen para el diseño estructurado de datos, diagramas de entidad relación, etc.

A mediados de los 70's, James Martin dio inicio al concepto de ciclo de vida para unificar y controlar todo el proceso de desarrollo de un sistema. Este método llamado Ingeniería de Información (IE) en el que se sigue un enfoque *top-down*, se identifican las necesidades de información durante la etapa de análisis con el fin de delimitar los alcances del sistema. Las metodologías *CASE (Computer Aided System Engineering)* fueron construidas siguiendo este enfoque.

Las Herramientas *CASE* empezaron a ser desarrolladas a principio de los 80's para soportar y automatizar algunas de las tareas encontradas en los métodos estructurados de desarrollo.

Algunos de los métodos más conocidos para el desarrollo de sistemas dentro de la ingeniaría de *software* se detallan a continuación.

Método Warnier/Orr

Warnier/Orr no es propiamente una metodología para la construcción de sistemas, existen los diagramas Warnier/Orr , el método de Warnier (lógica de estructura de datos, construcción de sistemas lógica, y ción lógica de programas) y el método de Orr (Desarrol stemas estructurados) están unidos pero estrictament ando no existe una metodología Warnier/Orr.

Es muy común que se confunda la metodología con los diagramas, tal vez esto sea porque los diagramas son la parte más visible de la mayoría de las metodologías; sin embargo, una metodología es mucho más que sólo diagramas y reglas de sintaxis.

Dentro del contexto de ingeniería de *software* un método es un procedimiento o una técnica que realiza una porción significante del ciclo de vida de un sistema. Una metodología es una colección de métodos basados en una filosofía común que juntos cubren parte o todo el ciclo de vida de un sistema.

El nombre correcto de lo que mucha gente llama metodología Warnier/Orr es: Desarrollo Estructurado de Sistemas de Datos (DSSD Data-Structured Systems Development) y es el resultado del trabajo conjunto de mucha gente.

En 1972, el artículo de Terry Baker "Chief Programmer Team Operations" en el semanario de sistemas IBM, reunió varias ideas de programación estructurada, diseño *Top-Down* e implementación, esto tuvo un gran impacto en el campo de la computación. Fue entonces cuando empezó la revolución estructurada en los Estados Unidos.

A principios de los 70's Ken Orr descubre que la estructura jerárquica de los programas es reflejada en la estructura jerárquica de los datos y fue por ese tiempo que Jean Warnier había hecho el mismo descubrimiento, pero además había construido una metodología sistemática alrededor de este descubrimiento.

La programación estructurada de los datos significa que pueden ser construidas soluciones predeciblemente correctas para una amplia gama de problemas de programación. La técnica de datos estructurados fue extendida para resolver programas complejos y arbitrarios.

Se desarrolló un esquema en el cual las entradas físicas son convertidas a entradas lógicas; las entradas lógicas se convierten en salidas lógicas; finalmente, las salidas lógicas son convertidas en salidas físicas.

Con este proceso de diseño de programas comienza la estrategia del diseño orientado a objetivos, éste inicia con la estructura de salida y trabaja hacia atrás, inicia con la salida lógica, entrada lógica y finalmente entrada física.

Poco a poco DSSD se movió de la metodología de diseño de programas a una metodología de diseño de sistemas. Después la metodología fue expandida para abarcar el diseño de bases de datos, definición de requerimientos y finalmente la planeación y arquitectura de los sistemas.

A nivel conceptual DSSD aún contiene rasgos característicos a nivel de programación. Por ejemplo, aún se enfoca (en su fase de diseño) a trabajos hacia atrás desde las salidas. DSSD trabaja hacia atrás hasta la base datos lógica y luego hacia las entradas. La base de datos lógica se transforma en una base de datos relacional y normalizada *(ver figura 1.1.1)*.

Mientras que una definición completa de los requerimientos (salidas más algoritmos) es un excelente punto de partida en el diseño, no es propiamente el punto desde el cual empezar la definición de los requerimientos. Así a través de los años, DSSD se

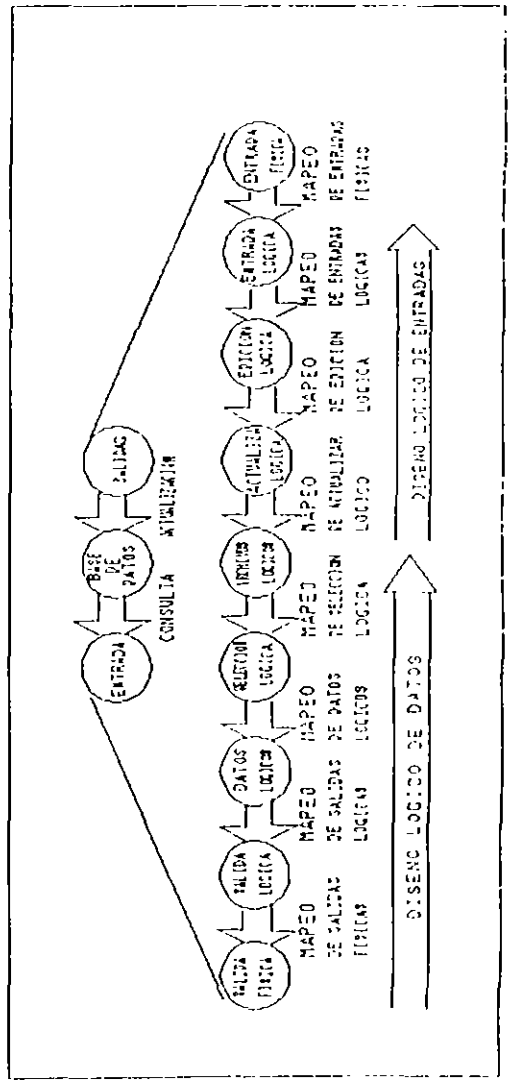


Figura 1.1.1 Trabajo hacia atrás desde la salidas del DSSD

ha extendido hasta cubrir primero el contexto, luego las funciones y finalmente los resultados del sistema.

Se hicieron necesarias un número de herramientas para facilitar este proceso. Los diagramas de Entidades ayudan a definir el contexto del sistema, y los diagramas de línea de montaje (una forma modificada de los diagramas Warnier/Orr) ayudan a definir el flujo funcional del sistema.

Las metodologías estructuradas de datos tienen ventaja sobre muchas metodologías orientadas a procesos, dado que son más rigurosas y por tanto proporcionan una mejor base para la verdadera integración a través de todo el ciclo de vida de un sistema. DSSD ha sido usado exitosamente en un gran rango de sistemas de *software*, desde sistemas comerciales en línea hasta sistemas de control de tiempo real. Cientos de personas lo han aprendido y cientos de sistemas han sido desarrollados con este método.

DSSD es un acercamiento a la ingeniería de *software* que ha proporcionado un marco de trabajo estable para incorporar nuevas tecnologías conforme éstas surgen. Por ejemplo, se han incorporado los prototipos de diseño en línea y en tiempo real sin sacrificar lo ya realizado.

Método Gane / Sarson

Este método está basado en el modelado lógico el cual consiste en tomar las ideas vagas necesarias acerca de los requerimientos y convertirlas en definiciones precisas tan pronto como sea posible, este proceso puede realizarse rápidamente si se cuenta con una técnica gráfica que nos permita tener la esencia del sistema sin ir al problema de la implementación física, con esto pudiera hacerse por ejemplo un p[illegible]totipo.

El modelado lógico puede verse como un proceso de 7 pasos:

1.- Desarrollo de un Diagrama de Flujo de Datos (DFD) general que describa lo que ocurre en cada área de la empresa. Para simplificar los diagramas se emplean solamente cuatro símbolos que producen un diagrama mostrando la naturaleza lógica de cualquier sistema de información a cualquier nivel deseado de detalle. Los DFD nos muestran los límites de una área dentro del sistema y de la empresa misma, los diagramas no son técnicos, es decir son fáciles de entender por cualquier persona familiarizada con el área descrita, nos muestran datos almacenados y los procesos que transforman esos datos.

2.- Derivar una lista de los datos que serán almacenados en cada almacén de datos, la lista puede afinarse revisando los datos de entrada/salida al sistema y determinando que representa cada dato.

3.- Revisar lo que el análisis entidad-relación pueda decirme acerca de la estructura de los datos. Obtener las Entidades y crear un diagrama con las entidades que se han definido, después buscar las relaciones existentes entre las entidades y dibujar líneas que muestren estas relaciones.

Por cada par de entidades se tiene una relación entre sus elementos, la relación puede ser uno a uno, muchos a uno o muchos a muchos. Estos diagramas nos dan mucha información acerca del sistema, mostrándonos todas las relaciones que existen entre las entidades involucradas *(ver figura 1.1.2)*.

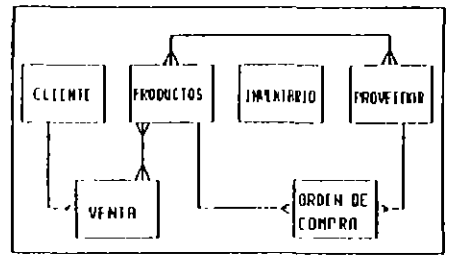


Figura 1.1.2 Todas las relaciones identificables entre entidades

4.- Emplear la información de los datos para describir un modelo que haga una liga entre dos tablas bidimensionales. Las tablas deberán estar normalizadas.

5.- Redibujar el DFD para reflejar una vista más precisa del sistema de datos como resultado del análisis del modelo entidad-relación y de la normalización.

6.- Particionar este modelo lógico de procesos y datos dentro de unidades de procedimientos, separar los procedimientos que puedan ser ejecutados manual o automáticamente.

7.- Especificar el detalle de cada unidad de procedimiento que se requiere para implementar el sistema: Extraer la unidad del DFD y mostrar donde queda dentro del sistema, detallar las tablas accesadas por esa unidad, mostrar un esquema de pantallas y reportes involucradas en la unidad y detallar la lógica y los procedimientos que serán implementados.

Una vez definidos los procedimientos se puede decidir si será un prototipo o se debe implementar directamente en algún lenguaje.

Los pasos 6 y 7 no son estrictamente hablando un modelado lógico dado que tratan de convertir el modelo lógico en modelo físico, sin embargo son parte del flujo natural a través del proceso que comienza en la definición de sistema y termina con el diseño físico.

Método Entidad-Relación

Es una metodología estructurada que puede sistemáticamente convertir los requerimientos del usuario en una base de datos bien diseñada. No es propiamente una metodología sino una aproximación.

Este método se basa en lo siguiente:

- Desarrollar un diagrama entidad-relación (ERD Entity Relational Diagram). Este paso identifica Entidades , Relaciones y atributos asociados además de la llave primaria de cada tipo de entidad.

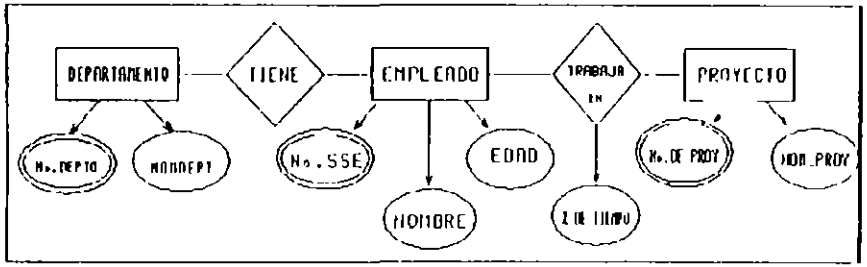


Figura 1.1.3
Diagrama Entidad Relación

Una entidad es una cosa, un concepto, una organización o un evento de interés para la organización que se está modelando. Una Entidad Tipo es una clasificación de entidades que satisfacen ciertos criterios. Una relación es una interacción entre entidades. Una relación tipo es una clasificación de relaciones basada en cierto criterio. Usualmente los sustantivos corresponden a las entidades mientras que los verbos corresponden a las relaciones.

El siguiente paso es identificar la cardinalidad de los tipos de relaciones, es decir con cuantos elementos de una entidad B se puede relacionar un elemento de una entidad A y viceversa. Se identifican después las propiedades (atributos) de cada Entidad-Relación y se expresaran gráficamente con círculos (o elipses). Las llaves primarias son identificadas con un doble círculo.

- Convertir los diagramas entidad-relación en archivos convencionales y estructuras de bases de datos. Existe un conjunto de reglas para hacer esto.

- Desarrollar los programas de aplicación basados en los archivos y estructuras de la base de datos. Si se está usando un Sistema Manejador de Bases de Datos (DBMS) es posible escribir en este momento un programa en SQL (System Query Language) para expresar la consulta a los datos.

El modelado Entidad-Relación puede ser usado no sólo como una herramienta de diseño sino como la base para el modelado en sistemas manejadores de bases de datos.

Método Yourdon

El método Yourdon es una colección genérica de ideas de ingeniería de *software* desarrolladas por muchas personas quienes han trabajado para la compañía Yourdon. Las ideas de todas estas personas se reunieron y se les dió el nombre de técnicas estructuradas: programación estructurada, diseño estructurado, y análisis estructurado, con esto Ed Yourdon intenta enlazar su metodología a las demás etapas del proceso de desarrollo de un sistema, para crear una metodología que compita con los métodos que sí cubren completamente el ciclo de vida de un sistema, es decir, los métodos que inician con la etapa de estrategia y análisis, diseño lógico y físico hasta la implementación, transición y producción.

Debido a la continua influencia de nuevas ideas de gente nueva, el método Yourdon está en continuo desarrollo, de tal forma que actualmente el método Yourdon es muy diferente del de hace 10 años, ahora se han incorporado las mejores ideas de diseño orientado a objetos y de análisis.

Este método consta de dos partes: herramientas y técnicas. Las herramientas son una variedad de diagramas gráficos usados para modelar los requerimientos y la arquitectura de un sistema de información. El más familiar de estos diagramas es el diagrama de flujo de datos (DFD), este diagrama ha sido extendido para soportar sistemas de tiempo-real. Los DFD incluyen control de flujo y control de procesos (*ver figura 1.1.4*).

Este método incluye además diagramas entidad-relación (ERD's) y diagramas de transición de estados (STD's State Transition Diagrams). Para terminar de definir los requerimientos de un sistema pueden emplearse las cartas de estructura para mostrar la organización de los módulos que serán implementados.

Para complementar la descripción del sistema es necesario un soporte textual adicional, un diccionario de datos que nos describa cómo está compuesto cada elemento y un conjunto de especificaciones de procesos que nos describa el comportamiento de los procesos en cada nivel.

La técnica del método Yourdon consta de guías que nos llevan de la nada a un modelo del sistema bien organizado. Esta técnica consiste en particionar (*top-down*) el sistema en funciones. Actualmente el método usa una técnica conocida como partición de eventos, inicia dibujando un diagrama de contexto el cual permite definir los alcances del sistema y las interfaces con fuentes externas. Después de las entrevistas al usuario se escriben una serie de eventos que ocurren en el medio ambiente externo y a las cuales el sistema responde, esto nos permite dibujar un primer diagrama de flujo de datos. Por ejemplo, si un sistema consta de 100 eventos tendrá 100 burbujas en el DFD, dado lo anterior es posible particionar los eventos en varios y unirlos en un solo proceso, esta técnica es parecida al diseño orientado a objetos.

El Diseño Estructurado Yourdon consta de 4 pasos:

1.- Diagramas de Flujo de Datos

Muestra los sistemas como un conjunto de procesos que transforman datos. Los diagramas de flujo de datos nos muestran la

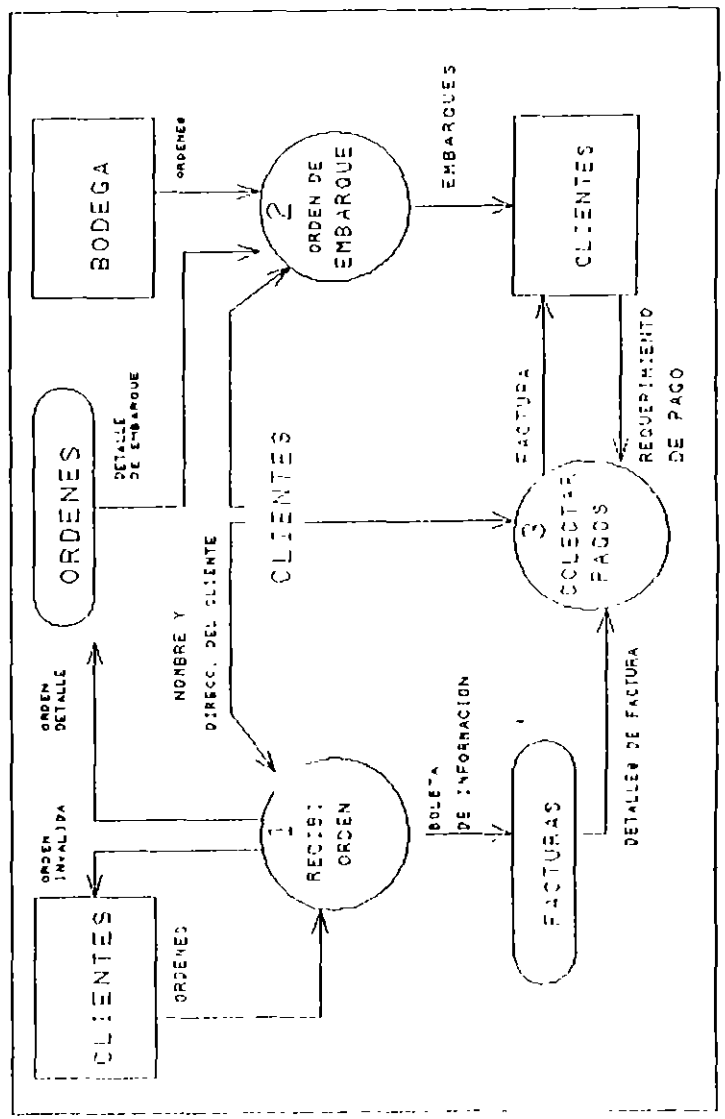


Figura 1.1.4 Diagrama de Flujo de Datos

relación entre los procesos y el flujo de información entre ellos. Los requerimientos en tiempo real son mostrados en los Diagramas de Flujo de datos vía el control de flujo y el control de procesos. Los comportamientos dependientes del tiempo son mostrados con diagramas de transición de estados y los datos involucrados son modelados de una forma simple en modelos de entidad.

2.- Cartas de Estructura

Las carta de estructura son una jerarquía de unidades procedurales que documentan el control del programa, el flujo de información entre programas y la lógica del programa. Se emplean las técnicas de Análisis de Transformación y de Análisis de Transacción.

3.- Evaluación del Diseño

En este paso se mide la calidad del diseño, para ello son empleados los criterios de acoplamiento y cohesión.

4.- Preparación para la Implementación

La lógica diseñada del programa es empaquetada en unidades de implementación física, llamadas unidades de carga.

Método de Ingeniería de Información (IEM)

El método de Ingeniería de Información sigue un enfoque *top-down*, desarrollando modelos en cada una de las etapas del ciclo de vida que cubre (estrategia, análisis y diseño), forzando el desarrollo de sistemas al manejo de metas y planes de estrategias.

IEM ofrece modelos de alto nivel para la representación total de una empresa durante la etapa de estrategia y diagramas de estructura de datos detallados (entidades, relaciones, atributos), descomposición funcional, flujo de datos y diagramas de dependencia, y definición de reportes y pantallas durante el "análisis de las necesidades de la empresa". Diagramas mostrando detalladamente la lógica del programa, incluyendo el manejo procedural y criterios de selección durante subsecuentes diseños y construcciones de sistemas. Matrices para el mapeo de las especificaciones del análisis y del diseño, para el manejo de metas, conjuntando los requerimientos de datos y procesos, y

asegurando una terminación e integración de éstos a nivel de aplicación.

Método DeMarco

DeMarco es un método de análisis estructurado orientado a procesos. Es utilizado para definir los requerimientos de un sistema en las etapas de análisis y diseño. Durante la etapa de análisis, DeMarco obtiene todo el conjunto de información manejada por los procesos actuales y los nuevos requerimientos de ésta y los utiliza para obtener la descripción de la funcionalidad del sistema a construir. Utiliza diagramas de flujo de datos como técnica de representación.

Pasos que sigue:

1.- Construir el modelo físico actual (DFDs).

2.- A partir del modelo físico, construir el modelo lógico actual (DFDs).

3.- Construir el modelo lógico del nuevo sistema a desarrollar incluyendo DFDs, definiciones del diccionario de datos, y especificaciones de los nuevos procesos.

4.- Crear una familia de nuevos modelos físicos (DFD).

5.- Obtener el costo y tiempos estimados para cada modelo.

6.- Seleccionar el modelo más adecuado.

Método Jackson

Jackson es una técnica de diseño orientada a datos, la cual se basa en las estructuras de datos requeridas para obtener la estructura de los procesos. Empieza asumiendo que la etapa de análisis ha sido terminada.

En este método el sistema es visto como un conjunto de procesos, los cuales se adecuan al manejo de las estructuras de datos, las cuales se consideran estáticas. Utiliza básicamente dos técnicas de diagramación: diagramas de redes, en los cuales se muestra el flujo de información a través de los procesos, y

diagramas arborescentes, en los cuales se muestran las relaciones jerárquicas entre procesos y datos.

Pasos que sigue.

1.- Dibujar los diagramas de estructura de árbol mostrando el flujo de datos entre cada proceso.

2.- Tomar todas las estructuras anteriores y formar un solo programa.

3.- Mostrar las operaciones necesarias para obtener la información de salida de la información de entrada y asignar cada una de éstas a los componentes del programa (procesos).

4.- Transcribir el programa a un texto estructurado, agregando tanto lógica de selección como ciclos.

CASE*Method

CASE*Method es una metodología que guía los proyectos hasta su terminación, a través de la aplicación de técnicas rigurosas para el desarrollo de sistemas. CASE*Method permite aprovechar los beneficios de la tecnología *CASE* dando un marco de trabajo estructurado para guiar y controlar el desarrollo productivo a través de todo el ciclo de vida.

Esta Metodología abarca el análisis estratégico, análisis detallado y diseño relacional de sistemas, permite llevar el control del sistema en todas las etapas de su ciclo de vida de una manera eficiente, ordenada y con la certeza de alcanzar los objetivos del sistema.

Durante las etapas de estrategia y análisis del ciclo de desarrollo del sistema, CASE*Method establece una comunicación más efectiva con los usuarios a través de técnicas, entrevistas y diagramación. Al entender los requerimientos de los usuarios desde el principio, se evitan revisiones costosas posteriores durante el desarrollo de sistemas.

Una vez identificados los requerimientos CASE*Method define el camino óptimo para completar estas tareas rápidamente y con precisión, basándose en la teoría relacional y extensos controles de calidad.

El método *CASE* es un método *top-down*, en el la gente está antes que los documentos, es un método de autochequeo, la implementacion es independiente de los requerimientos, está diseñado para el cambio, se implementa en forma metódica en todo tiempo.

CASE*Method soporta todas las etapas del ciclo de vida de un sistema.

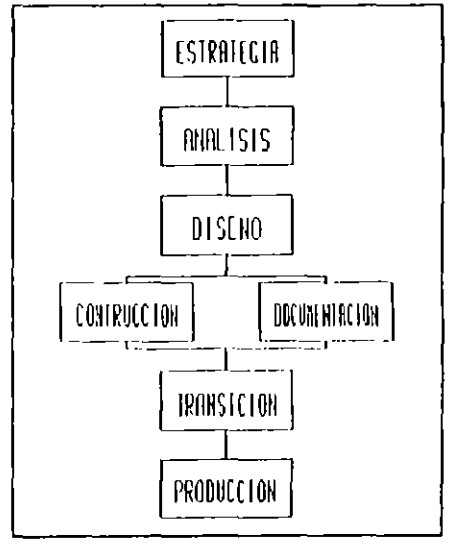


Figura 1.1.5
CASE*Method Enfoque estructurado para el desarrollo de sistemas

CASE*Method comienza con el principio del ciclo de vida, involucrando a usuarios y administradores en la especificación de los objetivos, requerimientos y prioridades que se necesitan para desarrollar el sistema.

Mientras se preparan e incrementan modelos detallados, pueden verificarse y confirmarse con los usuarios, creando prototipos

conceptuales del sistema completo. Utilizando el enfoque *top-down*, combinado con técnicas *bottom-up* para el chequeo cruzado, CASE*Method incrementa la calidad en cada etapa.

Después de completar el análisis, se crea el diseño físico y lógico para un desempeño y flexibilidad óptimos. La documentación e implementación del sistema ocurren paralelamente, mientras se construyen los sistemas y se realizan pruebas de unidad. La transición incluye el entrenamiento del usuario, la conversión de datos, y las pruebas del sistema, basado en una planeación anterior durante el análisis y diseño.

Etapa Estratégica

- Establecer los objetivos, prioridades y restricciones del negocio.
- Determinar los Modelos de Negocio.
- Establecer el compromiso administrativo, ganado a través de involucrar usuarios clave y secciones interactivas con retroalimentación.
- Realizar el plan de desarrollo y arquitectura del sistema.

Etapa de Análisis

- Especificación de los modelos detallados de la información necesaria para el sistema, vía los diagramas Entidad-Relación.
- Crear los modelos de los requerimientos funcionales utilizando técnicas de descomposición.
- Realizar diagramas de flujo de datos y matrices de chequeo cruzado.
- Realizar el plan de transición incluyendo la conversión y el entrenamiento.
- Definir la frontera del sistema.

Etapa de Diseño

- Establecer la arquitectura detallada del sistema.

- Transferir las especificaciones conceptuales a diseños físicos y lógicos.
- Realizar el diseño relacional de bases de datos incluyendo consideraciones de rendimiento y tamaño.
- De las especificaciones funcionales, realizar el diseño de módulos para sistema de aplicación.
- Realizar el diseño para la transición y la arquitectura de pruebas.
- Utilizar las matrices que ilustran referencias cruzadas de los componentes del sistema, para control de cambios.

Etapa de Construcción

- Creación de la base de datos relacional.
- Implementación de los sistemas de aplicación.
- Pruebas de unidad y revisiones del usuario.
- Aseguramiento de la calidad y aceptación.

Etapa de Documentación

- Documentación del sistema.
- Material de entrenamiento, operaciones y guías del usuario.

Etapa de Transición

- Conversión de los datos.
- Ejecutar las pruebas de simulación paralelas.
- Ejecutar las pruebas del sistema para verificar la funcionalidad, rendimiento, utilidad, recuperación, integridad y seguridad.
- Aceptación del usuario.
- Convertir el sistema para que sea completamente operacional.

Etapa de producción

- Realizar las pruebas de integridad.
- Ejecutar los procedimientos de soporte y monitoreo del sistema.
- Registro de los requerimientos.
- Cambio en los procedimientos de control.
- Realizar revisiones regulares.

ANALISIS COMPARATIVO DE LAS METODOLOGIAS DE DESARROLLO

Método Warnier/Orr (DSSD)

DSSD es un método enfocado a datos el cual contempla únicamente la etapa de diseño. Deriva las estructuras del programa de las estructuras de los datos siguiendo una estrategia *bottom-up*, pone atención en las salidas requeridas para obtener las estructuras del programa y las estructuras de los datos de entrada.

Pasos que sigue:

1.- Fase de planeación del proyecto: un plan de proyecto es desarrollado, conjuntando las metas de la empresa con un sistema particular de objetivos en un nivel de diseño lógico.

2.- Fase de definición de requerimientos: se definen en un nivel lógico y después en uno físico los requerimientos de usuarios y sistema, además de las necesidades de *hardware*. Se desarrollan diagramas de entidades (Diagramas de Flujo de Datos) describiendo las entidades y las transacciones que pasan a través de ellas. Las salidas de los procesos son mostrados utilizando diagramas de Warnier-Orr y un diccionario de datos.

3.- Fase de diseño: Se hace un diseño lógico de la base de datos normalizada y un diseño lógico de procesos, pasando después al diseño físico.

Ventajas

- Ofrece a los diseñadores pasos sencillos a seguir, usando una sola técnica de diagramación desde el principio hasta el final.
- Warnier/Orr es bueno para sistemas pequeños orientados a reportes.

Desventajas

- DSSD utiliza las salidas del sistema para obtener los requerimientos del mismo. Esta estrategia *bottom-up* puede causar un manejo incompleto o redundante tanto de datos como de procesos.
- DSSD sólo puede contemplar estructuras de datos jerárquicas, por lo que no se puede utilizar en el diseño de bases de datos relacionales ni distribuidas.

Método de Gane and Sarson

Gane and Sarson es un método *bottom-up* que contempla solamente las etapas de análisis y diseño.

Pasos que sigue:

1.- Construir el modelo lógico del sistema actual (DFDs).

2.- Construir el modelo lógico del sistema a desarrollar incluyendo especificaciones estructuradas (DFDs), un diccionario de datos y la nueva especificación de los procesos.

3.- Realizar un análisis entidad-relación.

4.- Diseñar las tablas de la base de datos que soporten al diseño lógico.

5.- Crear un nuevo modelo físico del sistema que consiste en redibujar el DFD para que muestre de una forma mas precisa los datos del sistema como resultado del análisis entidad-relación y de la normalización.

6.- Dividir el DFD en un conjunto de procesos.

7.- Especificar los detalles de implementación de cada uno de los procesos.

Ventajas

- Las convenciones que se utilizan para los diagramas de flujo de datos son las más rigurosas de todos los métodos.
- Los diagramas son fáciles de entender ya que sólo constan de cuatro símbolos.

Desventajas

- Es un método en el que puede omitir el análisis conceptual ya que empieza con una vista lógica del sistema existente con lo que no se podría satisfacer completamente las necesidades de los usuarios.
- Tiene un alcance limitado, ya que contempla de forma inadecuada las etapas siguientes a la etapa de diseño.

Método Yourdon

El Diseño estructurado Yourdon es un método que cubre la etapa del diseño durante el ciclo de vida de una sistema, fue desarrollado por Ed Yourdon a principios de los 70's. Es altamente recomendado para soluciones basadas en lenguajes de tercera generación y se enfoca al diseño de sistemas en tiempo real.

A finales de 1988 una revista de *software* publicó que el 30.2% de las empresas en Estados Unidos (la mayoría de gobierno, con necesidades de tiempo-real) emplean para el desarrollo de sus proyectos el método Yourdon, mientras que el 29.2% (la mayoría empresas comerciales) emplean métodos *top-down*.

El método Yourdon examina los sistemas actuales, obteniendo nuevas soluciones de los viejos sistemas. Este método es capaz de obtener soluciones para las necesidades de defensa y en general de cualquiera aplicación en tiempo-real.

Yourdon está bien adaptado para la etapa de diseño de los sistemas que requieren de procesamiento en tiempo real y se

requiere de una etapa de análisis extensa. Este método puede dirigir una funcionalidad compleja con solamente una lógica procedural limitada (3GL).

Ventajas

- Es ideal para aplicaciones de tiempo real.
- Tiene un diccionario de datos para la captura de detalles tales como atributos e identificadores únicos.
- No depende de los datos, dado que se concentra en la lógica para los programas.
- Yourdon es lo mejor para sistemas pequeños con requerimientos de procesamiento procedural y sistema de archivos simple.

Desventajas

- No ofrece una técnica propia para el modelado de entidades o diseño de datos, pero recomienda el uso de la técnica Chen de Entidad Relación.

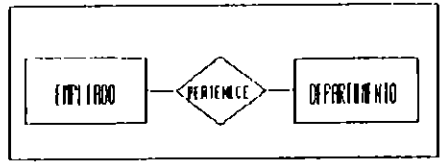


Figura 1.1.6 Modelo Entidad Relación de Chen

Esta técnica es muy superficial ya que no nos da mucha información acerca de las relaciones entre entidades. Esta forma de representación no es muy rigurosa y la normalización es imposible.

- No se crea un modelo de requerimientos de información conceptual.

- Sus diagramas no se prestan para describir los requerimientos del sistema, éstos son vistos únicamente como un mecanismo de procesos anticipados del sistema.

- No está diseñado para alternativas futuras, sus DFD's no están bien relacionados con las otras técnicas empleadas por Yourdon. No es posible saber en esta representación, donde toma lugar el proceso dentro de la organización. Yourdon no define el alcance de cada diagrama de flujo de datos.

- Es un conjunto de técnicas que no están bien acopladas. Las cartas de Estructura no corresponden a los Diagrams Entidad-Relación. No hay funciones que verifiquen el modelo de datos ni es posible verificar la lógica funcional del programa.

- El alcance limitado de Yourdon es su mayor desventaja ya que pasa por alto la etapa de análisis y posteriormente las etapas de desarrollo y se concentra en el diseño de programas procedurales.

Método de Ingeniería de Información (IEM)

El método de Ingeniería de Información cubre las etapas de estrategia, análisis y diseño del ciclo de vida de un sistema.

IEM ofrece diagramas de estructura de datos detallados (entidades, relaciones, atributos), descomposición funcional, flujo de datos y diagramas de dependencia, y definición de reportes y pantallas.

Ventajas

- Es un método que no está enfocado a procesos ni a datos, obtiene los requerimientos de información en un contexto balanceado entre necesidades de usuario y planes estratégicos.

- Los prototipos de entidades y modelado de funciones son hechas en conjunción con el usuario, para asegurar la satisfacción de éste.

Desventajas

- IEM no contempla el entrenamiento a usuarios, documentación, ni diseño en tiempo real. Las etapas de construcción y producción son tratadas en forma mínima.

- Los modelos de IEM son difíciles de leer.

Método DeMarco

DeMarco es un método de análisis estructurado orientado a procesos. Al igual que el método Gane and Sarson es utilizado sólamente en las etapas de análisis y diseño.

Ventajas

- DeMarco empieza el análisis utilizando los sistemas ya existentes, lo cual ayuda a los analistas y diseñadores a tener un punto de inicio.

- El modelado del sistema es iniciado con facilidad.

Desventajas

- DeMarco sólo contempla las etapas de análisis y diseño.

- Tiene un enfoque *bottom-up*, en el cual se diseña el nuevo sistema tomando las especificaciones de uno ya existente, teniendo como problema la posible pérdida de los nuevos requerimientos.

Método Jackson

Jackson es una técnica de diseño orientada a datos, la cual se basa en las estructuras de datos requeridas para obtener la estructura de los procesos, el sistema es visto como un conjunto de procesos. Utiliza básicamente dos técnicas de diagramación: diagramas de redes y diagramas arborescentes.

Ventajas

- Es útil cuando se tienen pequeñas necesidades de programación.

- Es capaz de contemplar requerimientos tanto a nivel sistema como a nivel proceso (módulo).

Desventajas

- Esta limitado al manejo secuencial de registros y al procesamiento en *batch* por lo que ignora la tecnología de bases de datos.
- Solo contempla la etapa de diseño.

CASE*Method

CASE*Method el un método que contempla todas las etapas del ciclo de vida. Mediante la utilización de este método se pueden conocer las necesidades de los usuarios, sin que éstas tengan que ser especificadas por sistemas existentes. Utiliza técnicas *bottom-up* como herramienta para asegurar la exactitud y terminación de una tarea.

Ventajas

- CASE*Method es flexible, ya que se puede adaptar con facilidad a proyectos de cualquier tamaño y complejidad.
- Enfatiza la utilización de referencias cruzadas con el propósito de detectar posibles errores, usando para ello matrices y diagramas de flujo de datos durante las etapas de análisis y diseño.
- Utiliza entidades conceptuales y el modelado de funciones para mostrar los requerimientos esenciales de la empresa, lo que se traduce a un diseño sin errores, es decir, se puede saber desde un inicio si el sistema va a satisfacer completamente las necesidades del usuario.
- Soporta el manejo de prototipos de interfaces con el usuario (pantallas, reportes, etc) durante las etapas de diseño y construcción.
- Sus técnicas de diagramación son fáciles de utilizar y entender.

- Ofrece sugerencias para la elaboración de documentación del sistema.

Desventajas

- No se puede aplicar el desarrollo de sistemas de tiempo real.
- Necesita de una técnica para especificar a detalle el diseño de los módulos.

Observaciones Finales

Después del estudio y análisis comparativo de diversas metodologías de desarrollo de sistemas, hemos podido observar que existen una gran diversidad de métodos para el desarrollo de sistemas.

Sin embargo y de acuerdo al estudio realizado, se observa que algunas metodologías ya han quedado obsoletas, pues se enfocan al desarrollo en lenguajes de tercera generación o bien son para sistemas de base de datos jerárquicas, muchas de éstas metodologías cubren sólo una parte del ciclo de vida de un sistema, es decir son incompletas, la mayoría de ellas se enfocan al diseño haciendo a un lado una de las partes más importantes como lo es el análisis. Algunos métodos emplean para sus modelos diagramas que son muy difíciles de entender, por lo cual se requiere de un entrenamiento especial para aplicarlos, otros métodos por el contrario emplean diagramas tan simples que por lo mismo dichos diagramas resultan incompletos.

Por otro lado basamos en la tendencia actual del mercado de *Software* hemos encontrado lo siguiente: una revista de *Software* publicó que hay un interés creciente en los métodos de ciclo de vida completo y *top-down*. De la misma manera en que la programación estructurada reemplazó al código no estructurado de los 70's así también en los 80's se muestra una adopción gradual de las técnicas estructuradas que cubren completamente el ciclo de vida de un sistema.

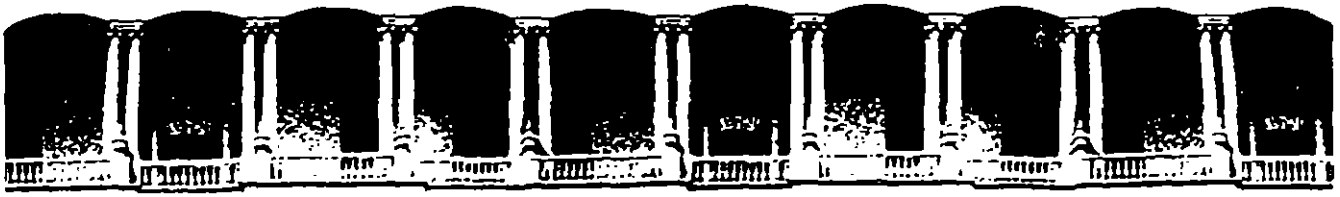
Las características del sistema que se desea implantar, se adaptan completamente a un modelo de bases de datos relacional y hemos podido observar con el estudio de las metodologías, que muchas de ellas no fueron pensadas para este tipo de implementación.

Se necesita implantar un sistema de información compartida, que sea sencillo de aprender por los usuarios finales y que incluya una documentación adecuada, además de que contemple modificaciones futuras conforme la empresa crezca.

Para el caso de estudio de esta tesis se necesita un sistema de bases de datos relacional y se empleará para este desarrollo los productos basados en la metodología *CASE*.

El Método de Desarrollo de Sistemas **CASE*Method**, es una metodología completa y moderna enfocada a la tecnología de bases de datos relacional que cubre completamente el ciclo de vida de un sistema, nos brinda una estrategia de planeación de sistemas de muy alto nivel, nos permite realizar un análisis detallado del sistema. Cuenta con técnicas de referencia cruzada que aseguran que el modelo sea completo y exacto. Sus DFD's son más rigurosos que otros métodos. Se basa en la comunicación entre usuarios y desarrolladores en el diseño de aplicaciones de 4a. generación y estados de post-diseño.

El método *CASE* es un método *TOP-DOWN*, es un método de autochequeo, la implementación es independiente de los requerimientos - tanto de *hardware* como de *software*, está diseñado para el cambio, se implementa en forma metódica en todo tiempo.



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

LA HERRAMIENTA "CASE"

EN EL DESARROLLO DE SISTEMAS

DIRECTORIO DE PROFESORES

ING. LAURA SANDOVAL MONTAÑO
JEFE DEL DEPARTAMENTO DE SERVICIOS ACADEMICOS
UNIDAD DE SERVICIOS DE COMPUTO ACADEMICO
FACULTAD DE INGENIERIA, UNAM
EDIFICIO DIVISION DE CIENCIAS BASICAS
CIUDAD UNIVERSITARIA, MEXICO, D.F.
TEL: 622 09 51 ó 622 81 00

'pmc.

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

LA HERRAMIENTA "CASE" EN EL DESARROLLO DE SISTEMAS

MATERIAL DIDACTICO COMPLEMENTARIO

MARZO-ABRIL 1995

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. Tel.: 521-40-20 Apdo Postal M 2265

TABLE OF CONTENTS

INTRODUCTION

LESSON 1 STARTING TO USE EASYCASE PLUS

LESSON 2 CREATING A CHART

LESSON 3 MODIFYING A CHART

LESSON 4 OTHER CHART FILE OPERATIONS

LESSON 5 DESCRIBING CHART OBJECTS TO THE DATA DICTIONARY

LESSON 6 LINKING CHART OBJECTS VIA THE DATA DICTIONARY

INTRODUCTION

Welcome to EasyCASE Plus

Welcome to EasyCASE Plus version 3.1, a powerful yet easy-to-use, easy-to-learn Computer Aided Software Engineering (CASE) tool. EasyCASE Plus is a graphical software environment designed to make the software development process more efficient and consistent. EasyCASE Plus features a consistent, industry standard IBM SAA/CUA compliant graphical user interface (GUI), similar in appearance and usage to the Microsoft Windows and OS/2 user interfaces, and a dBASE III Plus compatible data dictionary.

You can use EasyCASE Plus to perform structured analysis, structured design, and data and information modeling. EasyCASE Plus supports the following types of diagrams:

- Data flow diagrams (DFDs)
- Transformation graphs (TRGs)
- State transition diagrams (STDs)
- Structure charts (STCs)
- Entity relationship diagrams (ERDs)
- Data structure diagrams (DSDs)
- Data model diagrams (DMDs)
- Entity life history diagrams (ELHs)
- Logical data structure diagrams (LDSs)

With EasyCASE Plus you can create charts according to these methodologies:

- Yourdon/DeMarco
- Gane & Sarson
- SSADM versions 3 and 4
- Ward & Mellor
- Hatley-Pirbhai
- Yourdon/Constantine
- Chen
- Martin
- Jackson
- Bachman
- Elmasri & Navathe
- Shlaer & Mellor

Why Use EasyCASE Plus?

EasyCASE Plus automates the analysis and design phase of your project's development, eliminating some of the more mechanical, repetitive tasks so you can focus on design logic and concepts. EasyCASE Plus also provides support for real-time systems development and data and information modeling.

EasyCASE Plus 3.1, with its easy-to-use graphical user interface, makes it easy to select and manipulate objects on charts. The pulldown menus give quick access to commands without having to guess at, or remember, command names. The scroll bars enable you to access portions of the chart that are not currently visible. The dialog boxes allow you to select options, enter information, etc.

The integrated dBASE III Plus format data dictionary makes it possible to achieve a high level of consistency, because once you've described an object in the data dictionary, you can reuse the object, plus any relationships it may have, over and over, on the same or different diagram types.

You can link objects and charts in a hierarchy to provide functional decomposition. Once the objects or charts are linked, you can easily move back and forth between the various levels. You can also link text files to objects, and edit them using your preferred text editor. In addition, you can link chart objects to records, elements, and control tables to define data structures.

EasyCASE Plus lets you document your system as you proceed with the design. You can access text files, export diagrams into a word processor or desktop publishing program, export the data dictionary to a database, create and print reports and output the charts directly to a printer, plotter, or file.

How EasyCASE Plus Works

You use the chart editor (EASYCASE.EXE) to create, save, load, modify, and print charts. You describe objects on charts to the data dictionary from within the chart editor, and link the charts (and the objects on them) with other charts to create a hierarchical system of diagrams. The chart editor is completely integrated with other parts of the product. You can access the other portions of the product via the Utility menu. You can also access other external programs (such as your word processor or database) via the Utility menu.

You use the Data Dictionary and Reports Manager (DDRM.EXE) to add, modify, delete, copy, and rename dictionary entries. You can also browse data dictionary entries, export and import entries, and pack and sort the dictionary. You can also merge data dictionaries from other projects and/or other users. You can also print predefined chart and data dictionary reports to check the consistency and logic of your work.

The Analysis Manager (ANALYZE.EXE) is an optional, add-on module, included with EasyCASE Professional and the EasyCASE Plus Developer's Edition, that you use to verify the consistency and logic of your charts and data dictionary, and to print chart verification and level balancing reports.

The Schema Generator (SCHEMA.EXE) is an optional, add-on module, included with the EasyCASE Plus Developer's Edition, that allows you to produce database schemas in dBASE and/or SQL format, from entity relationship diagrams and the data dictionary.

All parts of EasyCASE Plus are integrated and accessible from other parts of the program. You can also access any of the modules directly from the DOS prompt.

About This Tutorial

This tutorial provides basic information on using the product, enables you to practice with it, and gives you hints and suggestions on how to proceed with your own work. It is designed to help you get familiar with EasyCASE Plus and its most frequently used features. It is aimed at the beginning user who has little or no experience with CASE tools. It does NOT attempt to teach you about structured analysis, design, and data modeling techniques or methods.

With EasyCASE Plus, you will find there are sometimes several ways to complete a task. This book does not contain complete details on everything you can do with EasyCASE Plus. See the Chart Editor User's Guide for more information on the tasks you practice here.

For a comprehensive tutorial about the usage of the chart editor to create an ERD and populate the data dictionary with database structure definitions, please refer to Appendix 1 of the Schema Generator User's Guide. That tutorial will take you through the process of defining a database using an ERD and then using the Schema Generator to create dBASE and SQL schemas.

The EasyCASE Plus Package

The EasyCASE Plus package includes the following items:

- Chart Editor User's Guide
- Installation Guide
- This Beginner's Tutorial
- Data Dictionary and Reports Manager (DDRM) User Guide
- Methodology Guide
- EasyCASE Plus disk pack with registration card
- Analysis Manager User Reference (if you purchased EasyCASE Professional)
- Schema Generator User's Guide (if you purchased EasyCASE Plus Developer's Edition)

To get started, you should use these items in the following order:

1. The first thing you should do is to install the EasyCASE Plus software on your PC. Read the Installation Guide first, and follow the directions to install the software.

NOTE: If you want to use this tutorial, you need to install the Sample Project files.

2. Read this tutorial and go through the practices and lessons to learn the basics about the product and to become familiar with how the menus, commands, and chart editor work.

3. Once you are familiar with the product, you will know which commands and features you would like more information about. Refer to the other documentation to get complete information about EasyCASE Plus.

If you are an experienced user and have used previous versions of EasyCASE Plus, read the Upgrade chapter in the Installation Guide for information on what has changed and how to upgrade your prior version projects. If you want to become familiar with the new EasyCASE Plus interface, go through Lesson 1 in this book and Chapter 2 of the Chart Editor User's Guide.

How To Use This Tutorial

This tutorial is organized so that you start by learning basic tasks and skills, and then advance to more complex procedures.

- This Introduction provides information about this and the other manuals in the EasyCASE Plus package.

- The Basic Tutorial (Lessons 1 to 4) contains information on starting EasyCASE Plus and using menus, commands, and dialog boxes; plus guided practices in scrolling, creating a chart, modifying it, options, printing, and saving.

- The Intermediate Tutorial (Lessons 5 and 6) contains information on identifying objects by accessing the data dictionary, and on linking objects to other charts, text files, records and elements.

For each lesson, you will see a short description of what is to be covered in the lesson. Next, the steps for the lesson are listed. At the end of each lesson are

suggestions, alternatives, and extra information that you can try when you do your own work.

Symbols and Conventions

To help you locate information easily, these symbols and conventions are used throughout the Tutorial:

- Examples of what you should type appear as:

  CD \ECPLUS

- Information you should note appears as:

- NOTE: Important or supporting information looks like this.

- Key names appear in small capital letters: ESC.

- Keys that should be pressed at the same time appear with + between them: CTRL+S.

- Bulleted lists (such as this one) give you information or choices.

- Numbered lists indicate procedural steps.

Terminology

Throughout the Tutorial these terms are used consistently to indicate operations:

Click. You click the mouse by pressing and releasing a mouse button. To select an object you would click the left mouse button while the mouse pointer is over the object.

Double Click. You press and release the left mouse button twice in quick succession.

Select. You select commands and objects on a chart. When you select a menu or dialog box command, an operation is executed. When you select an object or group of objects on a chart, the object or objects is/are ready for a subsequent operation, and EasyCASE Plus performs the next command you select on the selected object(s). To select a command or object, position the mouse pointer over it and then press and release (click) the left mouse button.

Drag. You move the objects on a chart with the mouse. To move an object, select the object, then press and hold down the left mouse button until the mouse cursor turns into a hand icon. Use the mouse to move the object to the desired position on the chart. Release the mouse button to place the object. The same process applies to dragging selection boxes around.

Enter. You enter information. To enter something, you type the information then press the ENTER key.

Cancel. To cancel a command or operation, press the ESC key.

Tutorial Files

There are Tutorial files in the Samples subdirectory that you will need to access for these lessons. These files are:

- ECPCONTX.DFD
- MODIFY.DFD

Basic Skills Tutorial

The Basic Skills section of the Tutorial introduces the EasyCASE Plus user interface and the fundamental skills you will need. The lessons are as follows:

Lesson 1	Starting to Use EasyCASE Plus
Lesson 2	Creating a Chart
Lesson 3	Modifying a Chart
Lesson 4	Other Chart File Operations

The material in each lesson builds on what you have learned in preceding lessons.

Intermediate Skills Tutorial

The Intermediate Skills section of the Tutorial takes you through identifying chart objects, describing them to the data dictionary, and linking them to other objects as follows:

Lesson 5 Interacting With The Data Dictionary

Lesson 6 Linking Chart Objects via the Data Dictionary

The material in each lesson builds on what you have learned in preceding lessons.

Schema Generator Tutorial

The Schema Generator Tutorial is provided in the Schema Generator User's Guide and takes you through the design of a sample database. It familiarizes you with creating an Entity Relationship Diagram (ERD), interacting with the data dictionary, linking chart objects to records, and defining databases using Records and Elements. It also covers the generation of SQL and dBASE III database schemas using the schema generator product.

LESSON 1
STARTING TO USE EASYCASE PLUS

Overview

This lesson covers:

- Running EasyCASE Plus and loading a sample chart
- Choosing commands from the pull-down menus
- Choosing options and supplying information via pop-up dialog boxes
- Selecting chart objects using the mouse
- Scrolling and zooming to vary the display of the chart in the chart window
- Exiting EasyCASE Plus

Before you start this lesson, you should have already installed EasyCASE Plus. If you have not, see Chapters 1 and 2 of the Installation Guide for information on how to do this.

NOTE: Keep in mind that EasyCASE Plus is designed to be very flexible, and that there are often many ways to accomplish your task. At the end of each lesson, you'll find information about alternative procedures.

This lesson should take you about 45 minutes to complete.

Running EasyCASE Plus

You will begin EasyCASE Plus by loading a chart from a sample project that is provided. In the next section, you will use this chart to explore the EasyCASE Plus user interface.

The following procedures assume you have installed EasyCASE Plus in the default (C:\ECPLUS31) directory on your hard drive, and the sample project in the SAMPLES subdirectory. If you have installed the product into a different

directory than the default, substitute the names of the directories you have created for \ECPLUS31 and SAMPLES.

To start EasyCASE Plus:

1. At the DOS prompt type:

   CD \ECPLUS31

   to go to the directory containing EasyCASE Plus.

2. At the EasyCASE Plus program directory, type:

   TSRINT

   This loads the memory-resident program TSRINT. It is a small utility task switching program, enabling you to suspend the EasyCASE Plus charting program and run EasyCASE Plus' adjunct programs (such as the Data Dictionary & Reports Manager), as well as external programs or DOS commands.

3. At the EasyCASE Plus program directory, type:

   EASYCASE SAMPLES ECPCONTX.DFD

NOTE: ECPCONTX.DFD is a sample data flow diagram used for the EasyCASE Plus tutorial.

4. The User Name dialog is then displayed.

5. Type in your name (up to 24 characters), and click on the 'OK' button.

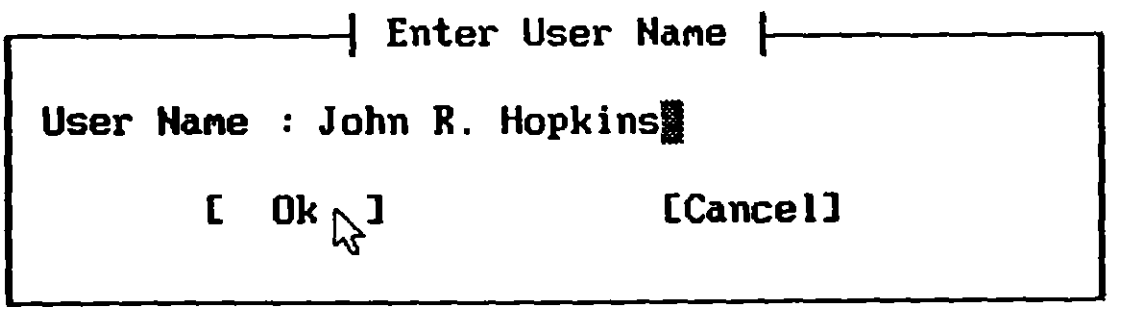


Figure 1.1 User Name Text Entry Box

NOTE: If this is the first time you have run the product after installing it, you will first be *required* to enter your (registered user) name, company and product serial number. Refer to the Installation Guide for further information about how to do this.

A message indicating that the program is loading the data dictionary for the project is displayed. The sample data flow diagram, ECPCONTX.DFD, is then automatically loaded and displayed in the chart window.

For more information on running EasyCASE Plus, refer to Chapter 3 of the Installation Guide.

The EasyCASE Plus User Interface

The EasyCASE Plus user interface is designed for clear, consistent and efficient use of commands and objects. If you have used other windowing applications, (such as Microsoft Windows 3.0) you will find familiar elements here: pull-down menus, icons, shortcut keys, scroll bars, and pop-up dialog boxes.

Take a few minutes to familiarize yourself with the screen layout:

The sample chart that you loaded (ECPCONTX.DFD) should now be displayed in the chart window. This chart is larger than the chart window to help illustrate some of the display tools. For the remainder of this lesson, you will use this chart to become accustomed to the EasyCASE Plus user interface.

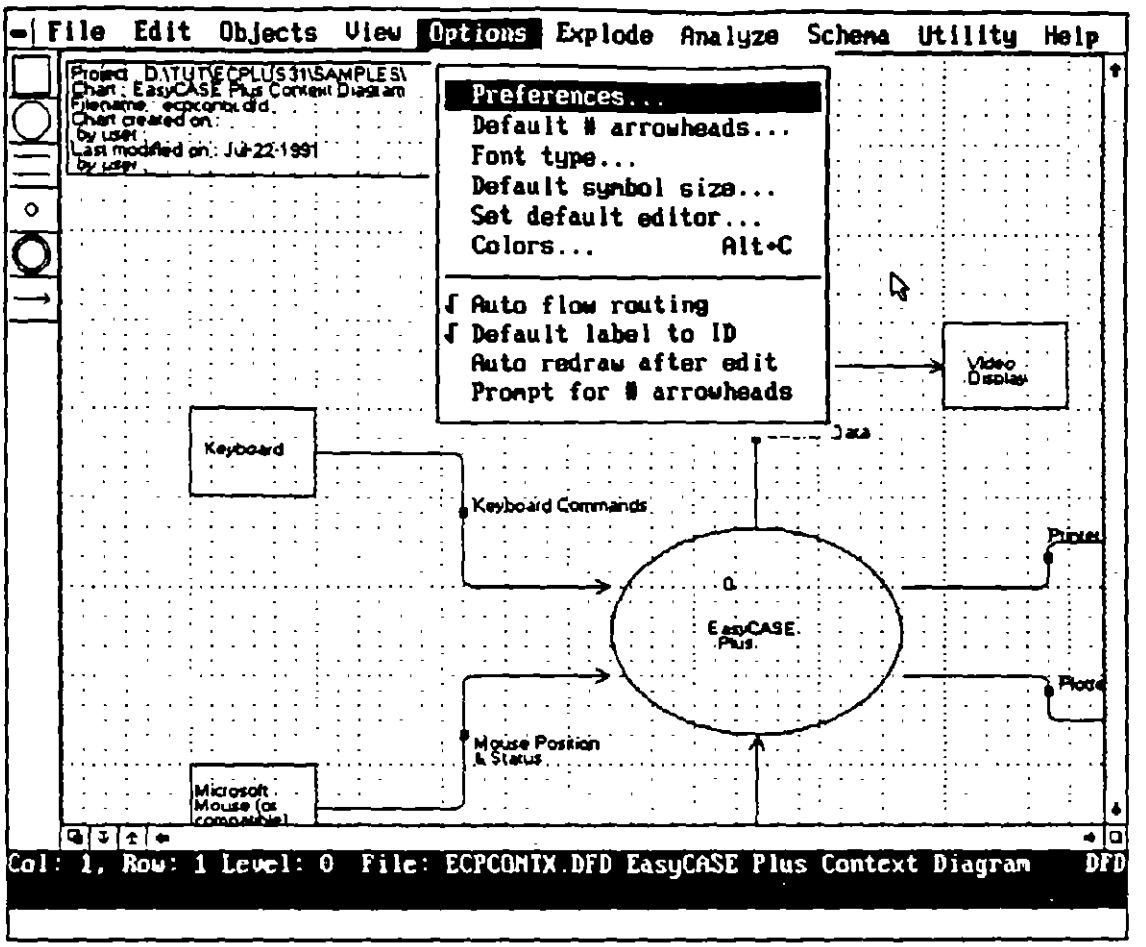


Figure 1.2 The EasyCASE Plus Chart Editor User Interface

The Chart Window

A chart contains a maximum of 32 rows and columns roughly equivalent to 6 sheets of regular U.S. letter size paper (3 wide by 2 high). The chart window displays an 8 column by 9 row section of the entire chart.

Menu Bar

The Menu bar (located at the top of the screen) contains the ten pull-down menu options. These are described in more detail in a subsequent section.

Drawing Grid

The drawing grid segments the chart into positions that symbols can occupy. Each 'box' on the grid can hold a symbol which is centered in that box. The chart is a 32 by 32 grid of these boxes. The outline of each box is composed of finely spaced dots. All other objects that you place on a chart (connections, couples, labels and text blocks) will "snap" (automatically move) to locations indicated by the more widely spaced dots.

Scroll Bars

The horizontal and vertical scroll bars make it easy to move around large diagrams.

Scroll Arrows. The scroll arrows move the chart one row (up and down arrows) or column (left and right arrows) at a time. You can increase this scroll increment to 4, 8, or 16 by using the CTRL, ALT and SHIFT keys, respectively.

Scroll box. The Scroll box moves the position of the chart window relative to the entire chart, when you drag it along the scroll bar.

Status Line

The Status line contains the chart offset coordinates, the current level number (in a linked chart set), the filename, descriptive name, and type of the current chart.

Prompt Line

The Prompt line, located below the status line, displays status and error messages, or prompts you for your next action.

Redraw Icon Button

This icon, located in the lower right corner of the screen, refreshes the chart window. Use this if 'debris' has been left behind in the chart window as a result of adding, deleting, or moving chart objects or text.

Quick Zoom Icon Button

This icon, located in the lower left corner of the screen, is a toggle switch that zooms out to display the full chart in the chart window when you click on it, and then returns to actual size when you click on it again.

Explode Up and Down Icon Buttons

These icons, located in the lower left corner of the screen, let you move between a chart object and a chart, text file, record, control table, or element that it has been previously linked to.

Exit Icon Button

This icon, located in the upper left corner of the screen, allows you to quit the application. You will be prompted to save the chart if any changes have been made. You will also be prompted to save your preferred option settings.

Chart Object Icons

These icons, located down the left side of the screen, display the object types available for the current chart type. The object types displayed here correspond to those available in the Objects menu. You can select an object type to place either by using the Chart icons, or from an entry in the Objects menu.

The Menus

The ten pull-down menus located in the menu bar across the top of the screen list all of the EasyCASE Plus commands available as follows:

- **File menu** commands are for accessing projects; loading, merging, saving and exporting chart files; printing; and exiting the program.
- **Edit menu** commands allow you to copy, move, modify, delete, label, and identify objects within the chart window. They also allow you to select a group of chart objects to manipulate.
- **Objects menu** commands allow you to select symbols and connections for placement on a chart.
- **View menu** commands affect the display of the entire chart, or objects on the chart. They also allow you to zoom in and out of areas of the chart.
- **Options menu** commands let you set the appearance of the user interface, and the characteristics and defaults to use when creating a chart and adding objects to it.
- **Explode menu** commands allow you to move between linked objects and charts.

- **Analyze menu** commands allow you to invoke Analysis Manager functions, if installed, including chart rule checking and level balancing.

- **Schema menu** commands allow you to invoke Schema Generator functions, if installed, including selecting the schema type and viewing schema files.

- **Utility menu** commands allow you to access the supporting EasyCASE Plus programs, as well as DOS and external programs.

- **Help menu** commands provide quick access information to enable you to use EasyCASE Plus effectively.

The individual components of the user interface ar)w you use them are described in Chapters 2 and 3 of the Chart Editor U .r's Guide. Chapter 2 describes how to access commands from the pull-down menus and pop-up dialog boxes. Chapter 3 describes how to scroll and zoom the chart using the scrollbars, mouse and keyboard.

Setup Chart Editor Options

Prior to beginning to work through the lessons in this tutorial, please take a moment to make sure your chart editor options are setup exactly as follows. If they are not, you may not get exactly what is described in the tutorial steps.

View Menu

Drawing grid	On
Display title block	On
Double 1 bar on ERDs	On
Directed connects	Off
Open arrowheads	On
Filled arrowheads	Off
Rounded corners	On

Options Menu

Default # arrowheads	One way single
Font type	Helvetica
Default symbol size	Normal
Auto flow routing	On
Default label to ID	On
Auto redraw after edit	Off
Prompt for # arrowheads	Off

Preferences (via Options Menu)

Auto identify	Off
Auto label	Off
Allow place new label	On
Auto export to dBASE	Off
Auto export to SDF	Off
Auto export to HPGL	Off
Update label from DDE	Off
Flag deleted DDEs IDs	Off
Warning beep	On
Create backup chart	On
Use named charts	On

Explode Menu

Show explosions	On

NOTE: When a menu option is On in the View or Options menu, a checkmark appears to the left it. For Preferences, when an option is On, an 'X' appears to the left of it.

Getting Help

Whenever you are uncertain about the use of a command or procedure, try using the on-line Help facility. In many cases, you can find the information you need here much more quickly than by referring to the EasyCASE Plus Chart Editor User's Guide.

To access on-line Help:

1. Click on the Help menu option or press the F1 key.

   A pop-up dialog box lists the available help topics, as shown in Figure 1.3.

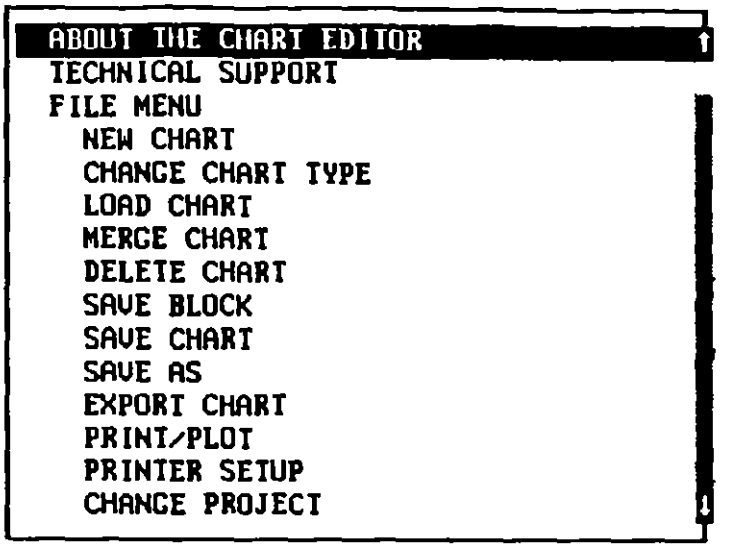


Figure 1.3 Help Topics List

2. Click on the 'Merge Chart' entry.

   A text box containing the help information for merging charts is displayed as shown in Figure 1.4. Since this information extends past the size of the text box, you can use the scroll bar at the right of the box to scroll forward and backward through the available help text.

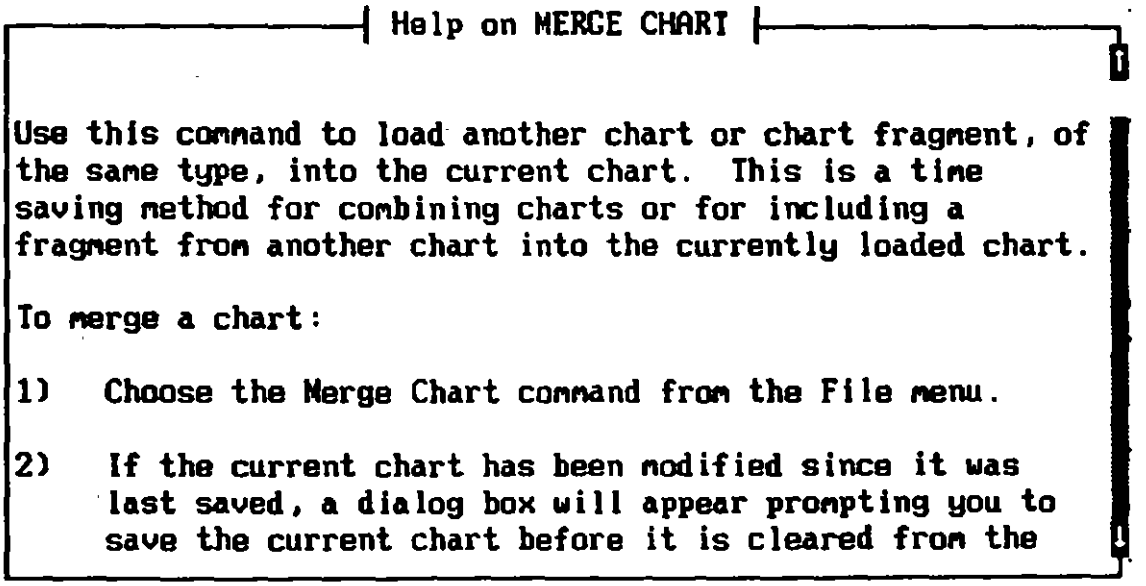


Figure 1.4 Help Screen for Merge Chart Topic

3. Press the ESC key or click the right mouse button to remove the help text box and return to the Help topics list.

4. Press the ESC key or click the right mouse button again to remove the help topic list box, and return to the chart.

Selecting Single Objects on the Chart

When you want to move, copy, delete, modify, or label an object, you need to first select the object. Chart objects are:

- Symbols
- Connections
- Labels
- Blocks
- Text

Selecting an object highlights it, and indicates to the chart editor that it is ready for a subsequent editing operation.

The following sections describe how to select a chart object. For additional information, refer to Chapter 5 of the Chart Editor User's Guide.

Selecting a Symbol

Selecting a symbol is as easy as simply clicking on it with the mouse. To select the symbol labeled 'EasyCASE Plus' on the chart:

1. Position the mouse pointer in the center of the symbol containing the text label 'EasyCASE Plus', as shown in Figure 1.5.

2. Click the left mouse button.

   The object, and the text inside it, are highlighted and surrounded by small boxes to indicate that it is selected.

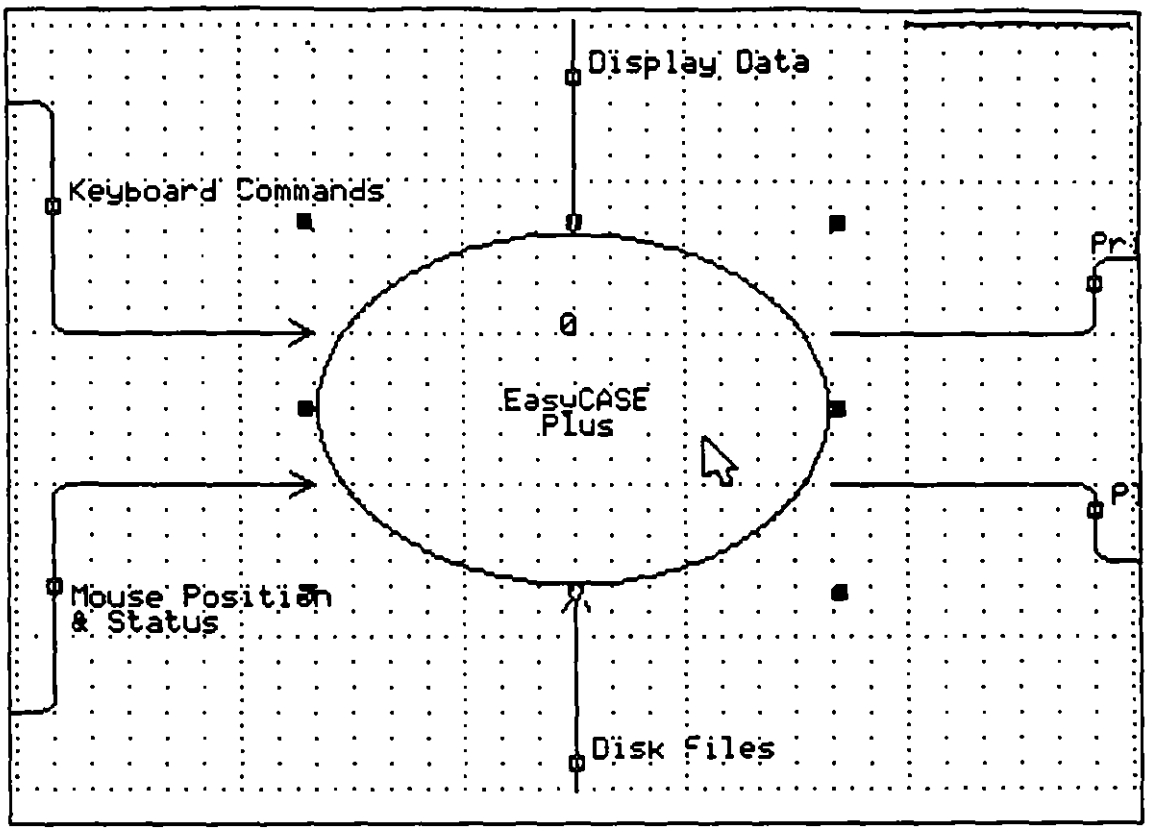


Figure 1.5 Selected a Symbol

To de-select the selected symbol:

Click the right mouse button, or click the left mouse button elsewhere on the screen (anywhere outside of the symbol).

The symbol is no longer selected and the highlight and marker boxes are removed.

Selecting a Connection

A small box is displayed midway along the middle line segment of each connection. This is the "handle" you will use to select the connection.

To select the connection labeled 'Display Data':

1. Position the mouse pointer over the small box located on the connection labeled 'Display Data' as shown in Figure 1.6.

2. Click the left mouse button.

The connection, and the adjacent label, are highlighted. Small boxes are displayed at the start, end, corners, and mid-points of the connection to indicate that it is selected, as shown in Figure 1.6.

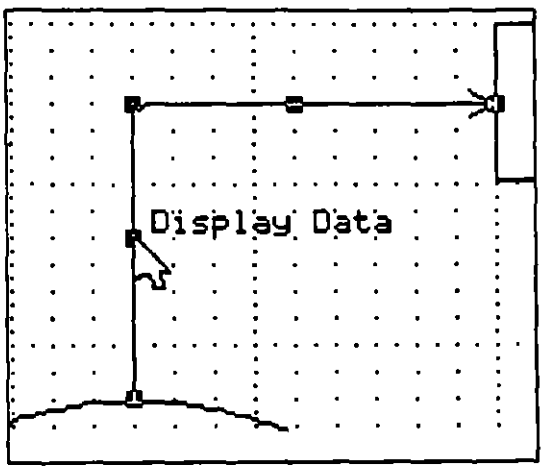


Figure 1.6 Selected a Connection

To de-select the connection:

Click the right mouse button, or click the left mouse button elsewhere on the screen.

The connection is no longer selected and the highlight and marker boxes are removed.

Selecting a Symbol Label

Labels can appear inside symbols, and are selected as follows:

To select the 'Video Display' symbol label:

1. Position the mouse pointer over the symbol labeled 'Video Display'.

2. Click the left mouse button.

   The symbol and its label are highlighted.

3. With the mouse pointer still positioned over the Video Display symbol, click the left mouse button again.

The 'Video Display' label alone is highlighted and enclosed by a dashed bounding box as shown in Figure 1.7. The label is now selected for a subsequent operation.

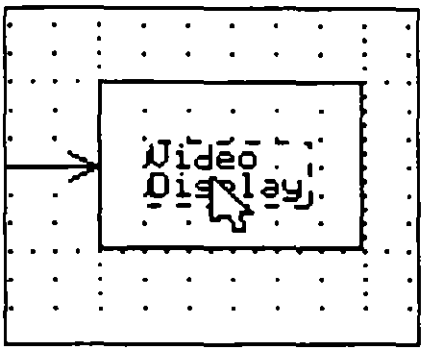


Figure 1.7 Selected a Symbol Label

Selecting a Connection Label

Labels can also appear next to a connection, and are selected as follows:

To select the 'Display Data' connection label:

1. Position the mouse pointer anywhere over the 'Display Data' text.

2. Click the left mouse button.

3. The 'Display Data' label is highlighted and enclosed by a dashed bounding box as shown in Figure 1.8. The label is now selected for a subsequent operation.

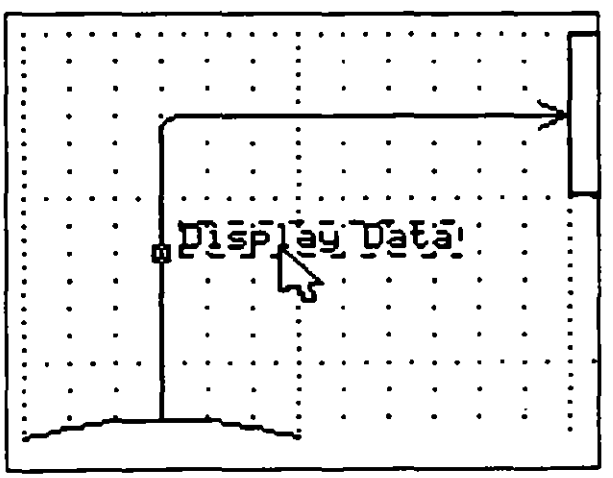


Figure 1.8 Selected a Flow Label

To de-select the label:

Click the right mouse button, or click the left mouse button elsewhere on the chart (anywhere outside of the label box).

Selecting a Data Couple

Data couples appear next to connections on structure charts. Before you can select a data couple you must first load a structure chart.

To load a structure chart:

1. Click on the 'File' menu to pull it down.

2. Click on the 'Change type' option. A list of available chart types will appear.

3. Click on 'Structure Chart'. A blank structure chart named UNTITLED.STC will appear in the chart window.

4. Click on the 'File' menu again to pull it down.

5. Click on the 'Load chart' option. A list of available structure charts will appear.

6. Click on the "Page-Jones Structure Chart" entry.

7. Click on 'OK'; the chart will load.

To select a data couple:

1. Locate the data couple 'Customer P.O. Data' on the connection between function symbol "1.0 Fill Customer Purchase Order", and function symbol "1.3 Add Customer P.O. Record". You may need to scroll the chart one click to the left to access it easily.

2. Position the mouse pointer over the small round base of the data couple.

3. Click the left mouse button.

   The couple and its label are highlighted and surrounded by small boxes that indicate it is selected ready for a subsequent editing operation, as shown in Figure 1.9.

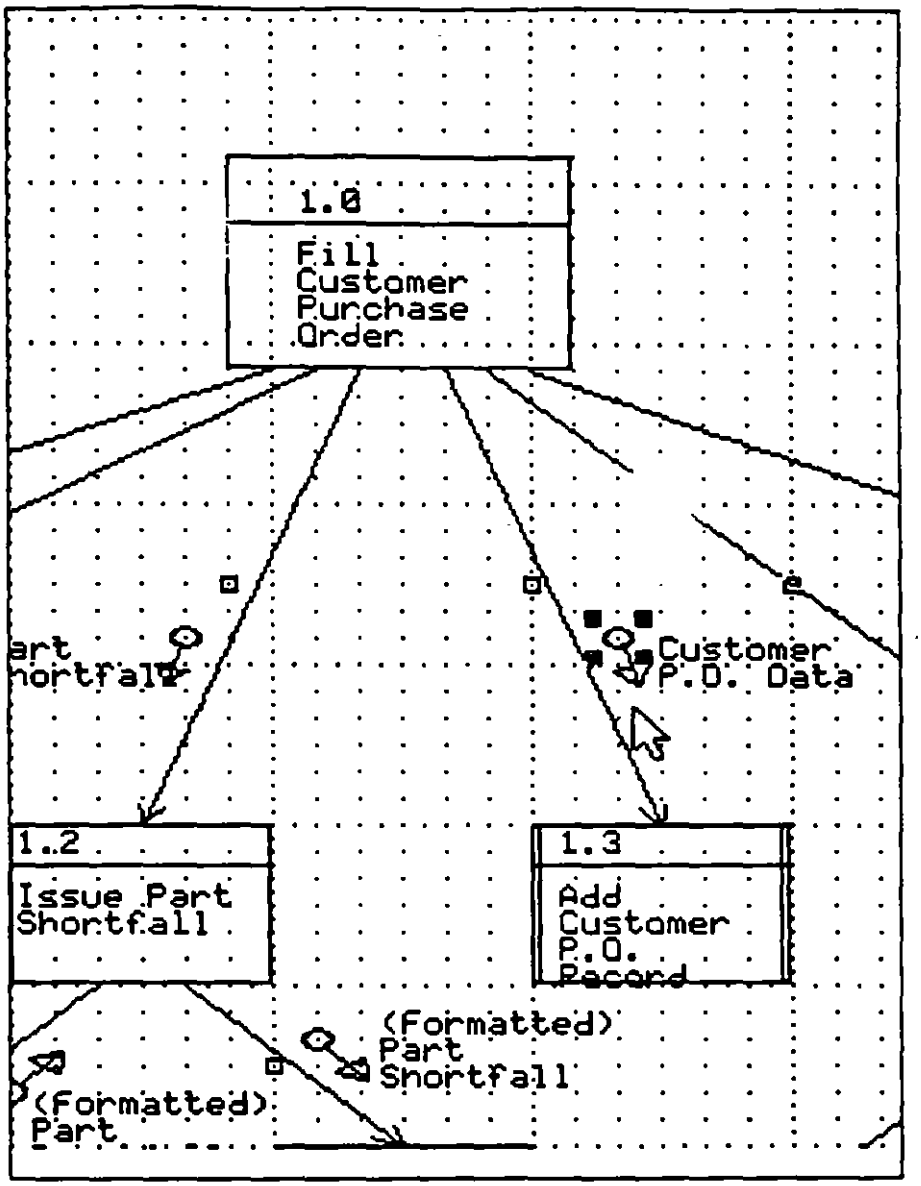


Figure 1.9 Selected a Data Couple

A couple label is selected in the same way as for a connection label by clicking on the label text.

To return to the EasyCASE Plus Context Diagram, follow the instructions in steps 1 through 6 above, but choose Data Flow Diagram (Yourdon symbol set) in step 3 and EasyCASE Plus Context Diagram (ECPCONTX.DFD) in step 5.

Selecting a Group of Objects

In addition to selecting single objects, you can select a number of objects together as a group or 'block'. Once selected, you can then perform the standard move, copy, delete or save operations, just as you would for single objects.

To select a group of objects visible in the chart window:

1. Position the mouse pointer just outside the upper left corner of the external entity symbol labeled 'Keyboard.'

2. Press and hold down the left mouse button. The mouse pointer will change into an icon in the form of a hand with a pointing finger.

NOTE: If you accidentally click on a symbol, just press ESC, or click the right mouse button, to cancel the operation.

3. While still holding down the left mouse button, move the pointer one grid square down and to the right, as shown in Figure 1.10.

   A dashed box will appear; the top left corner of which is located at the mouse pointer position when the left button was pressed.

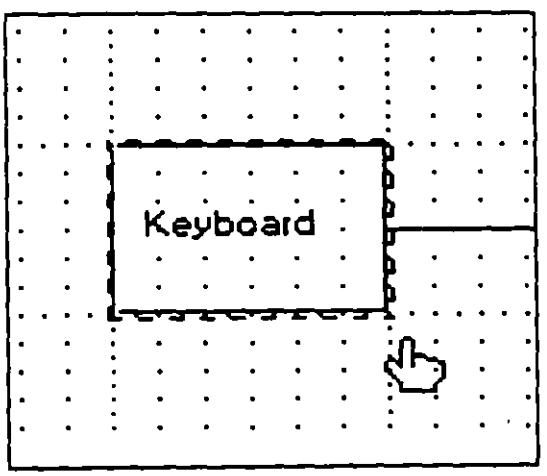


Figure 1.10 Drawing the Chart Object Group Selection Box

4. Still holding down the left mouse button, move the mouse pointer down and to the right of the symbol labeled 'Microsoft Mouse'. The bottom left corner of the dashed selection box will follow the mouse pointer.

The dashed box should encompass the 'Keyboard' and 'Microsoft Mouse' symbols, as well as their connections, as shown in Figure 1.11.

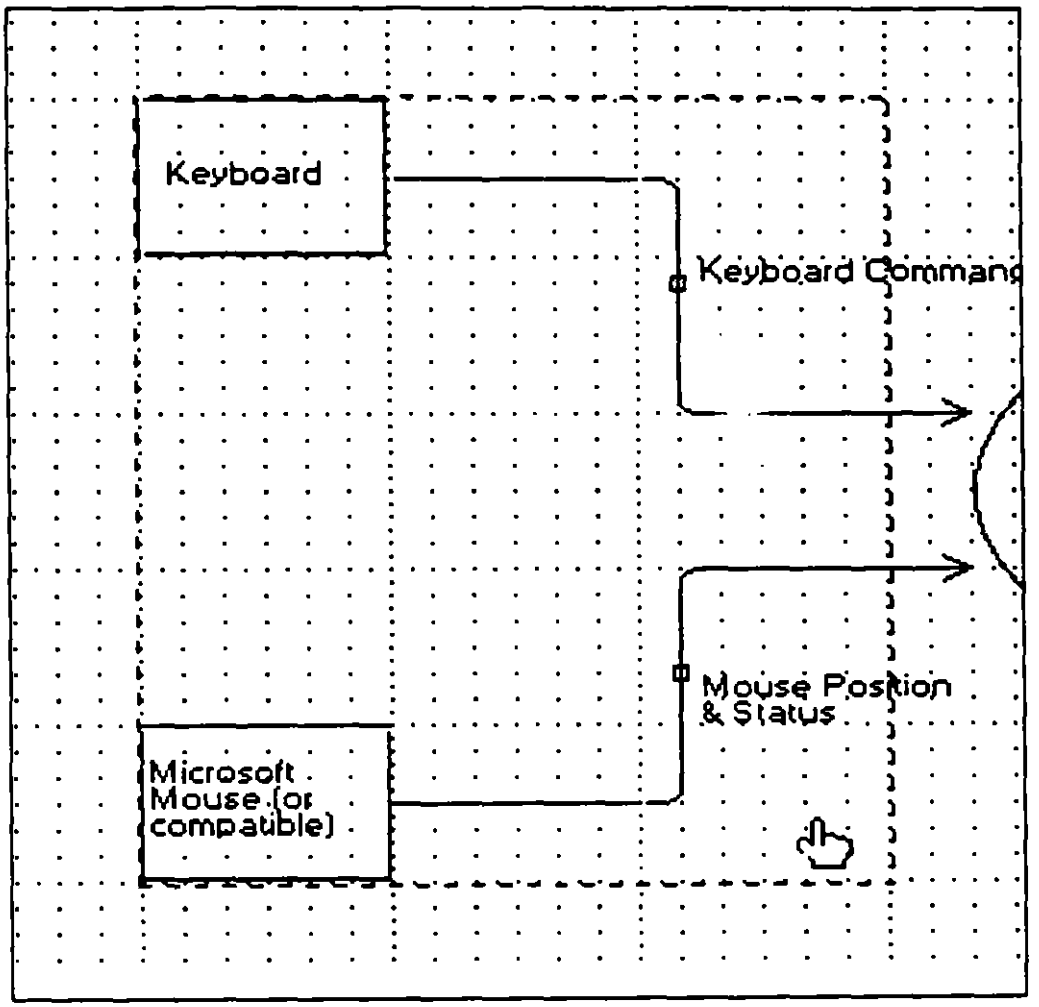


Figure 1.11 Selecting a Group of Chart Objects

5. Release the left mouse button.

   The block of objects is highlighted and enclosed by a dashed bounding box and is selected for a subsequent operation, such as move, copy, delete, or save block.

To de-select the group of selected chart objects:

Click the right mouse button or click elsewhere on the chart (outside of the selection box).

NOTE: When you also want to include objects in your selection that are outside of the current chart window (on the chart but not currently displayed), the chart window will scroll as you hold the left mouse button and enlarge the selection box as it contacts the edge of the chart window.

Scrolling

Scrolling makes it possible for you to see portions of the chart that are not currently displayed in the chart window. The chart drawing area is 32 rows by 32 columns. The chart window displays 9 rows by 8 columns, or approximately 25% of the entire chart.

The EasyCASE Plus screen has two scroll bars. The one down the right side of the screen scrolls the chart vertically. The one across the bottom of the screen scrolls the chart horizontally.

Scroll Arrow Buttons

The scroll arrows, located at each end of the scroll bars, move the chart one row or column at a time.

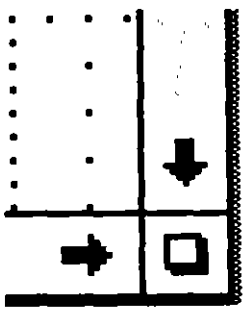


Figure 1.12 Right and Down Scroll Arrow Buttons

To scroll down a few rows:

Click on the down arrow, located at the bottom of the vertical scroll bar, several times. The chart window scrolls down the chart one row at a time.

To scroll right a few columns:

Click on the right arrow, located at the right end of the horizontal scroll bar, several times. The chart window scrolls right across the chart one column at a time.

NOTE: In both cases, you can increase the scroll increment to 4, 8 or 16 columns or rows at a time by holding down the CTRL, ALT, or SHIFT key respectively while clicking on the scroll arrow.

Scroll Boxes

The scroll box, located between the scroll arrows on each scroll bar, moves the chart window over the chart and allows for more rapid scrolling.

To scroll toward the bottom of the chart:

1. Position the mouse pointer over the scroll box on the vertical scroll bar.

2. Press and hold down the left mouse button.

   The scroll box is outlined by a dashed box. This outline follows the mouse pointer along the scroll bar as you move the mouse.

3. While holding down the left mouse button, drag the scroll box outline down to the end of the scroll bar.

4. Release the left mouse button.

   The bottom area of the chart is now displayed in the chart window.

You do the same type of operation with the scroll box on the horizontal scroll bar.

Zooming

The zoom feature enables you to quickly switch back and forth between macro and micro views of your chart. There are three zoom levels available ranging from actual size to a mode whereby you can see the entire chart on the screen.

You can zoom using any of the following approaches:

- Click both mouse buttons simultaneously
- Click on the Quick Zoom icon
- Select the 'Quick Zoom' option from the View Menu
- Press the CTRL+Z key combination

The quickest way to zoom is with the mouse buttons or Quick Zoom icon.

To zoom using the mouse buttons:

1. Press and hold down the right mouse button.

2. Then, press the left mouse button also.

3. Release both buttons simultaneously.

4. The full chart is reduced to appear in the chart window, as illustrated in Figure 1.14.

5. Repeat Steps 1 through 3 to zoom back to the actual size chart.

6. The actual size chart is re-displayed in the chart window.

NOTE: The position of the mouse pointer when you click the buttons determines what area of the chart will be displayed in the chart window when you zoom from full chart back to actual size display.

To zoom using the Quick Zoom icon:

1. Click on the Quick Zoom icon (located in the lower left corner of the chart window) as shown in Figure 1.13.

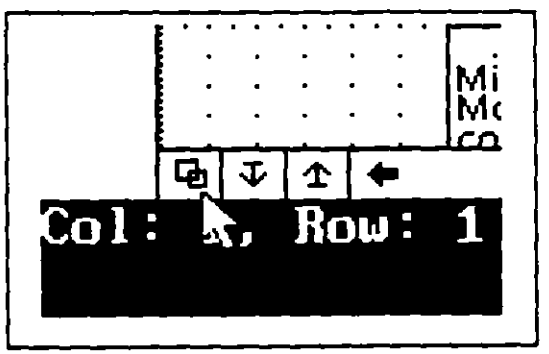


Figure 1.13 Quick Zoom Icon

2. The full chart appears in the chart window, as illustrated in Figure 1.14.

3. Click on the Quick Zoom icon again to zoom back to the Actual Size chart.

4. The Actual Size chart is then re-displayed in the chart window.

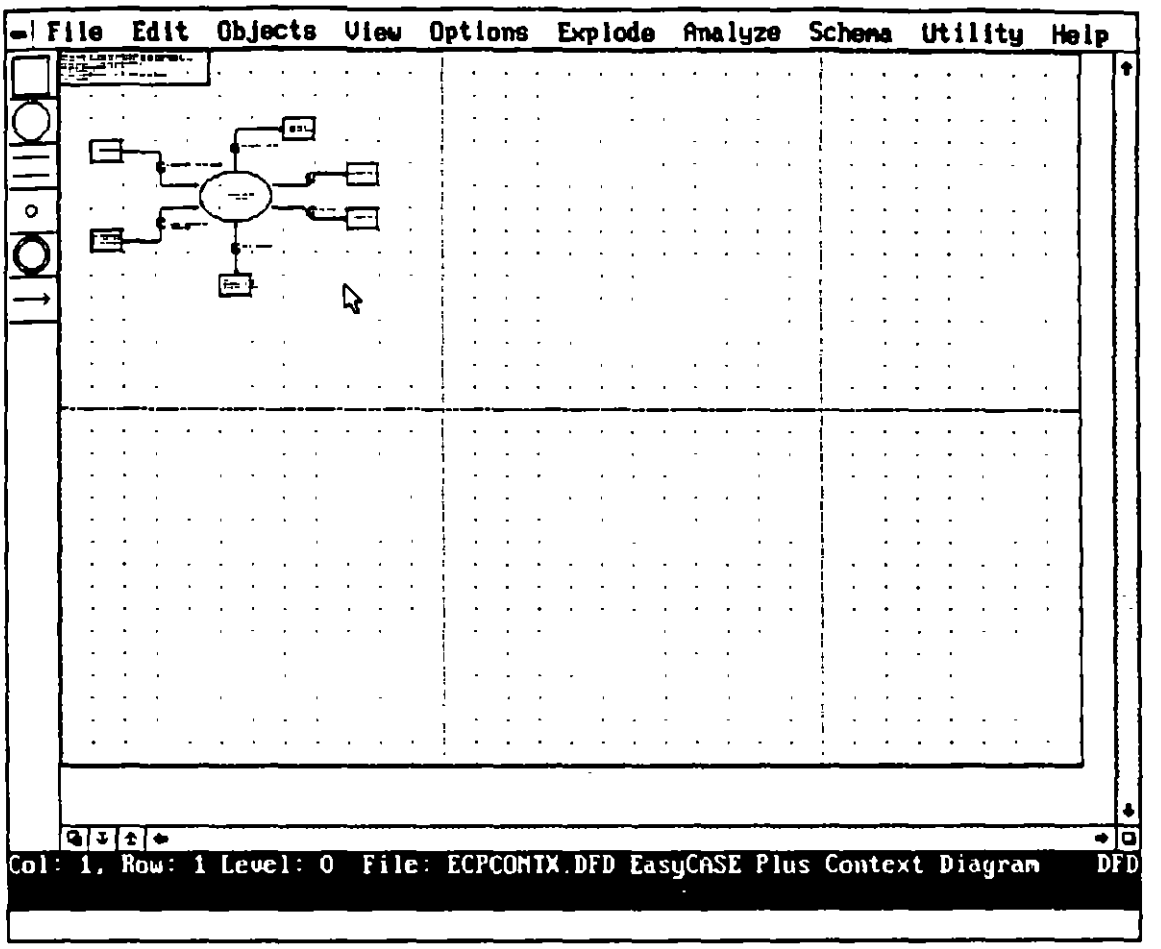


Figure 1.14 Zoomed (Full) Chart

NOTE: When you quick zoom from full chart to actual size using the Quick Zoom icon, CTRL + Z keys, or 'Quick Zoom' menu option, the chart window is centered on any selected object. If an object is not selected, you are returned to the position where you were located when you previously 'quick zoomed' from actual size to full chart mode.

Zooming Using the View Menu Commands

The View menu commands, as shown in Figure 1.15, give you greater flexibility in choosing which part of the chart you want to display when you zoom.

- The **Full Chart** command displays the entire chart with all objects in reduced size.

- The **Half Size** command displays only the half of the chart that you select. The objects are easier to see than in full chart mode.

- The **Actual Size** command displays objects at normal size, so you can see detail and work with the objects.

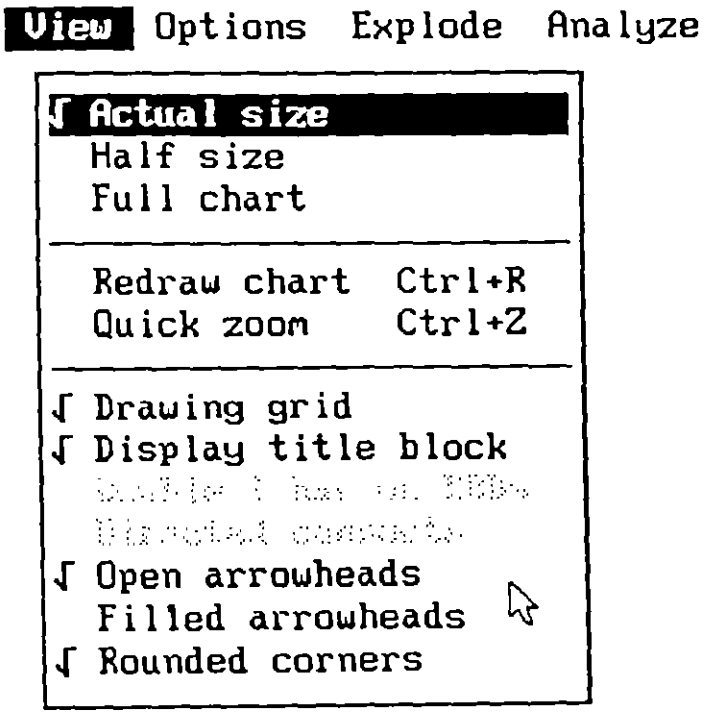


Figure 1.15 View Menu

Zooming between Actual Size and Full Chart

Your chart should be displayed at actual size, the size you used for creating and modifying the chart.

To zoom to the Full Chart display:

1. Select the View menu.
2. Select the Full Chart option.
3. The entire chart is then displayed in the chart window, as illustrated in Figure 1.16.

Use the same approach to move from Actual size to Half size view. While at Actual size view, choose the Half size option from the View menu.

To zoom back to Actual Size display:

1. Select the View menu.

2. Select the Actual Size option.

3. A selection box is displayed. The mouse pointer is positioned at the top left corner of this box.

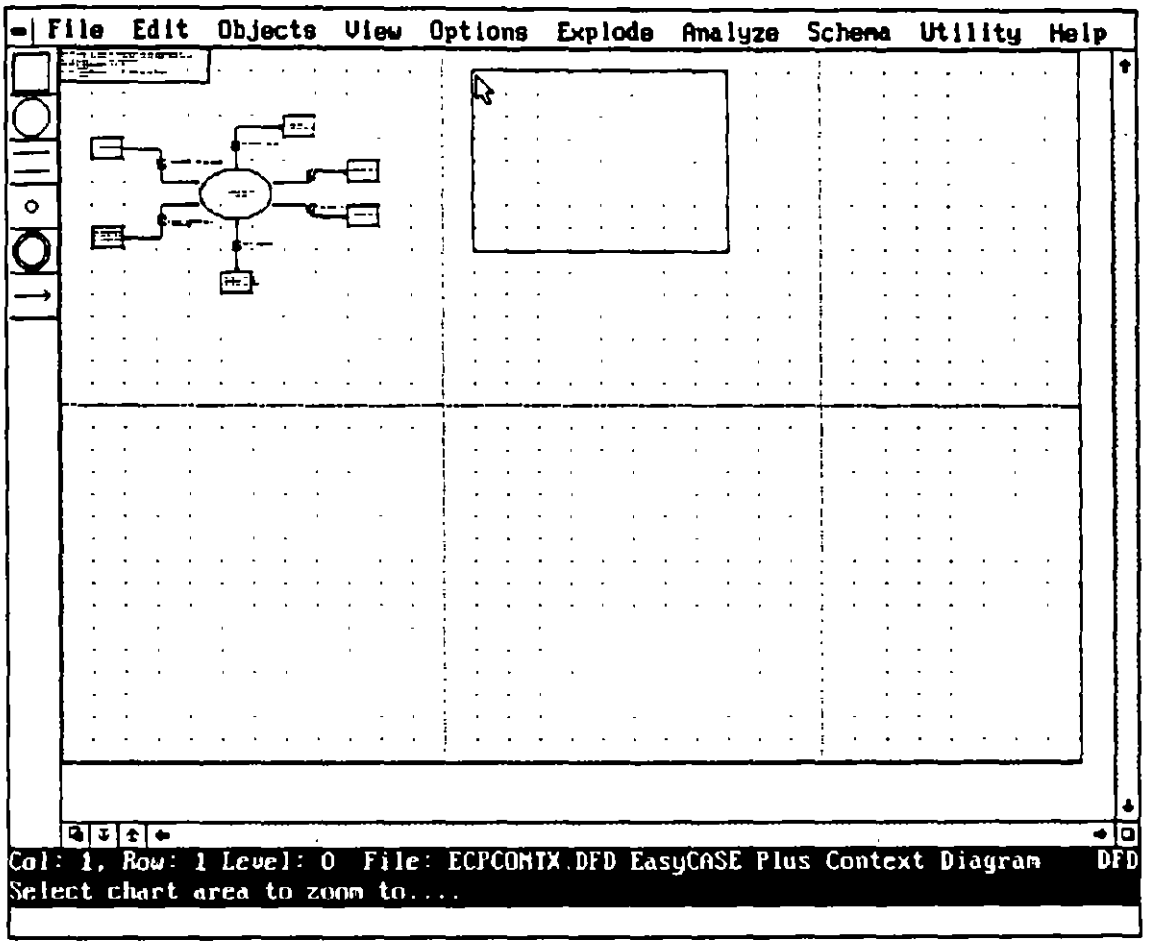


Figure 1.16 Actual Size Zoom Selection Box

4. Press and hold down the left mouse button.

5. Use the mouse to drag the selection box over the area of the chart that you want to display in the chart window at actual size.

6. Release the left mouse button.

7. You are prompted to verify that this is the selected area to zoom into.

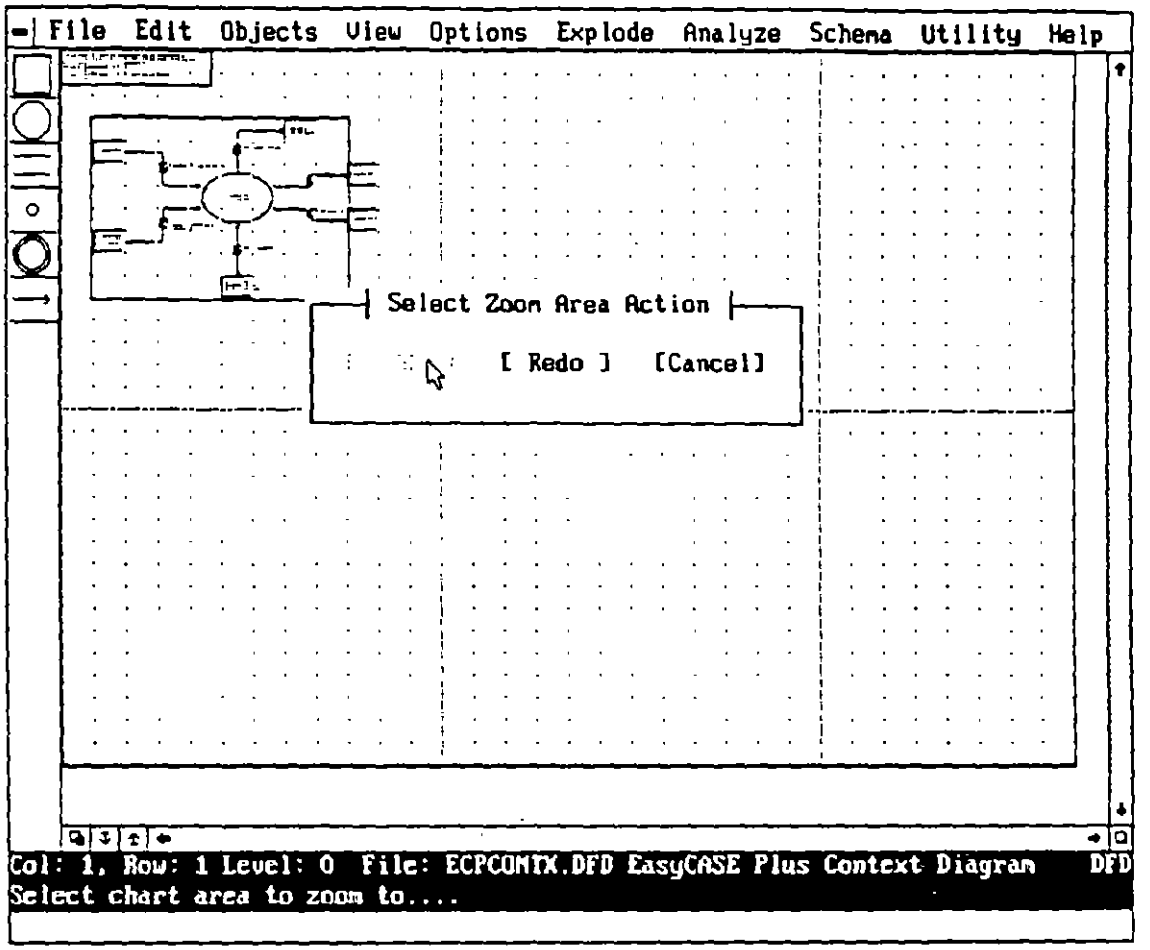


Figure 1.17 Prompt to Verify Zoom Action

8. Click on 'OK' to accept your choice.

NOTE: If you want to reposition the selection box, click on 'Redo' and reposition the selection box over the desired chart area. Click on 'Cancel' or press the ESC key to cancel the operation and remain at full chart view.

Zooming From Full Chart to Half Size

Set the zoom level so the chart is displayed in Full Chart mode.

To zoom to the Half Size display from the Full Chart mode:

1. Select the View menu.

2. Select the Half Size option.

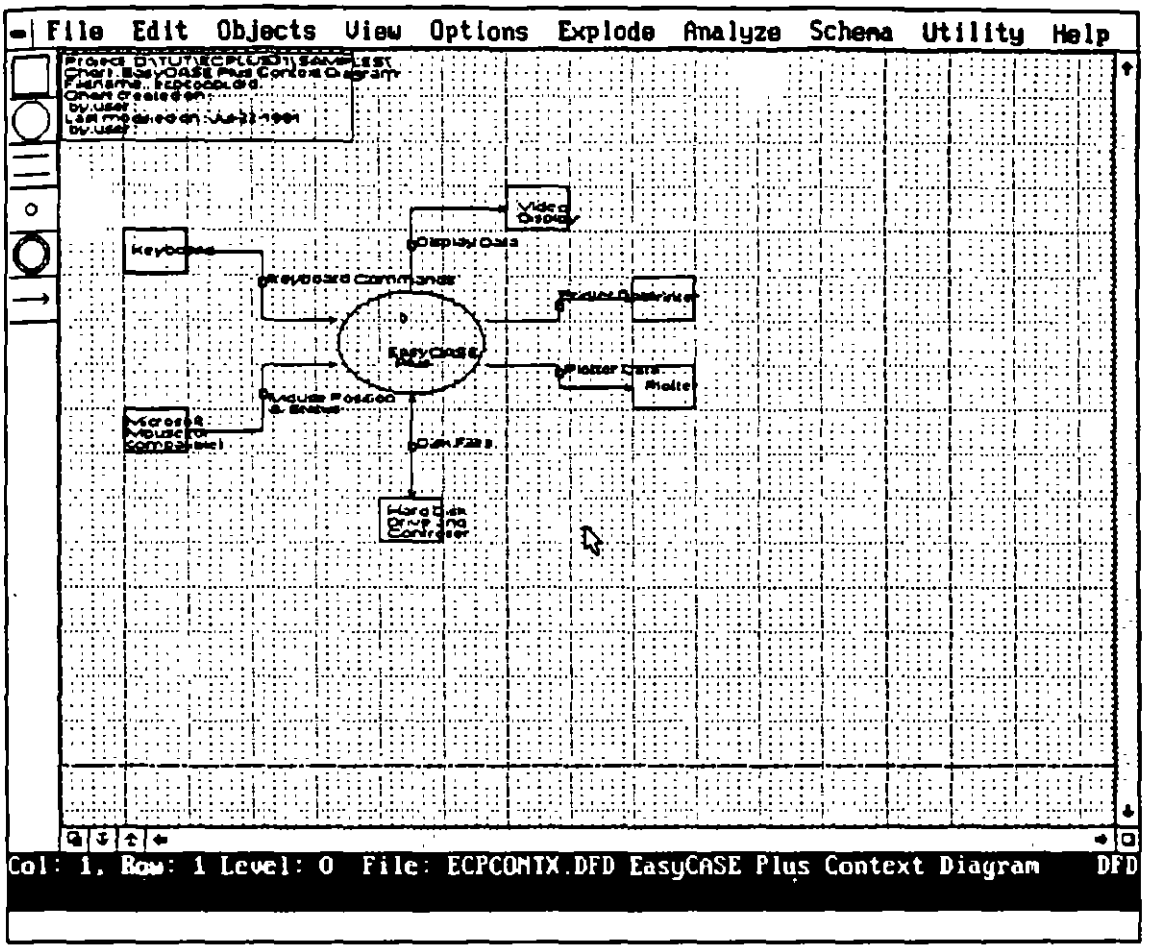


Figure 1.18 Chart Window Displayed in Half Size Mode

3. Use the same approach as described in steps 3 through 8 of the previous section to zoom from Full chart to Half size. The selection box is larger because more of the chart can be seen at Half size than Actual size.

Use the same procedure to move from Half size to Actual size view. While at Half size view, choose the Actual size option from the View menu.

Redrawing the Chart Window

You can redraw (refresh) the chart window at any time simply by clicking on the Redraw icon located at the lower right corner of the chart window (between the right and down scroll arrows), as shown in Figure 1.19.

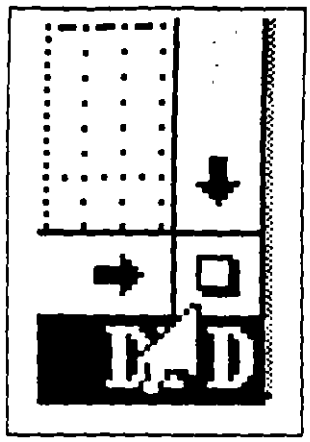


Figure 1.19 Redraw Icon

NOTE: You can also press the CTRL + R key combination, or choose the Redraw Chart command from the View menu, to redraw the chart.

Exiting EasyCASE Plus

To exit EasyCASE Plus and return to DOS, choose the 'Exit to DOS' command from the File menu. You are prompted to save any changes made to the current chart, if any.

To exit EasyCASE Plus:

1. Select the File menu.

2. Click on the 'Exit to DOS' option.

3. If you have changed the current chart and have yet to save it, you are prompted to save the changed file.

4. Click on the 'No' option to discard any changes made.

5. The exit dialog box is then displayed, prompting you to verify that you want to exit the program.

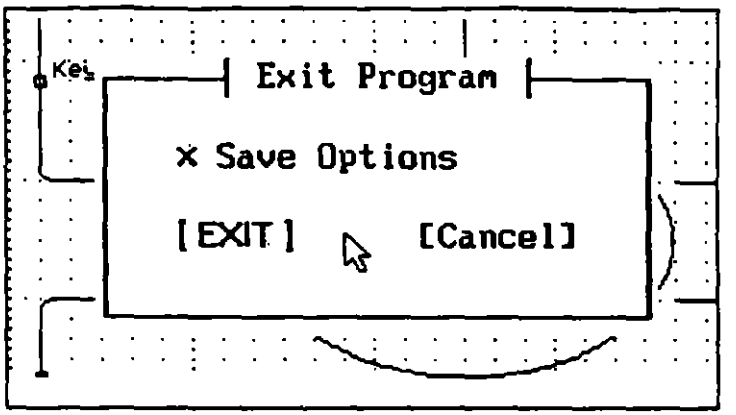


Figure 1.20 Exit Prompt

The 'Save Options' setting allows you to choose to save your current options and configuration (these options are available in the View and Options menus). By default, the 'Save Options' setting is turned On. Leave it at this setting.

6. Click on the 'Exit' option to exit the program.

   The EasyCASE Plus program is ended, and you are returned to the DOS prompt.

Alternatively, you can also exit EasyCASE Plus by clicking on the 'Exit' icon, as follows:

1. Position the mouse pointer over the 'Exit' icon button located at the top left corner of the screen, as shown in Figure 1.21.

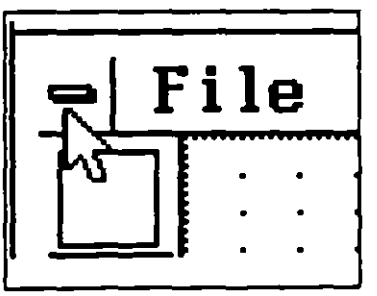


Figure 1.21 Exit Icon

2. Click the left mouse button on the icon.

3. You are then prompted to save the current chart and option settings, just as when you chose the 'Exit to DOS' command at the File menu.

You can also exit by pressing the CTRL + X key combination.

Alternatives

Starting EasyCASE Plus from the DOS Command Line

To display a list of projects:

- Use the */P* switch. This switch displays a listing of projects and directories after the welcome screen.

  The command is:

  EASYCASE /P

  This switch enables you to select an existing project directory, or enter the name of a new project directory. When the product runs, you should see a dialog box similar to Figure 1.22 from which you can access a project in a subdirectory or navigate through the drives and directory structure on your disk to a project directory located elsewhere.

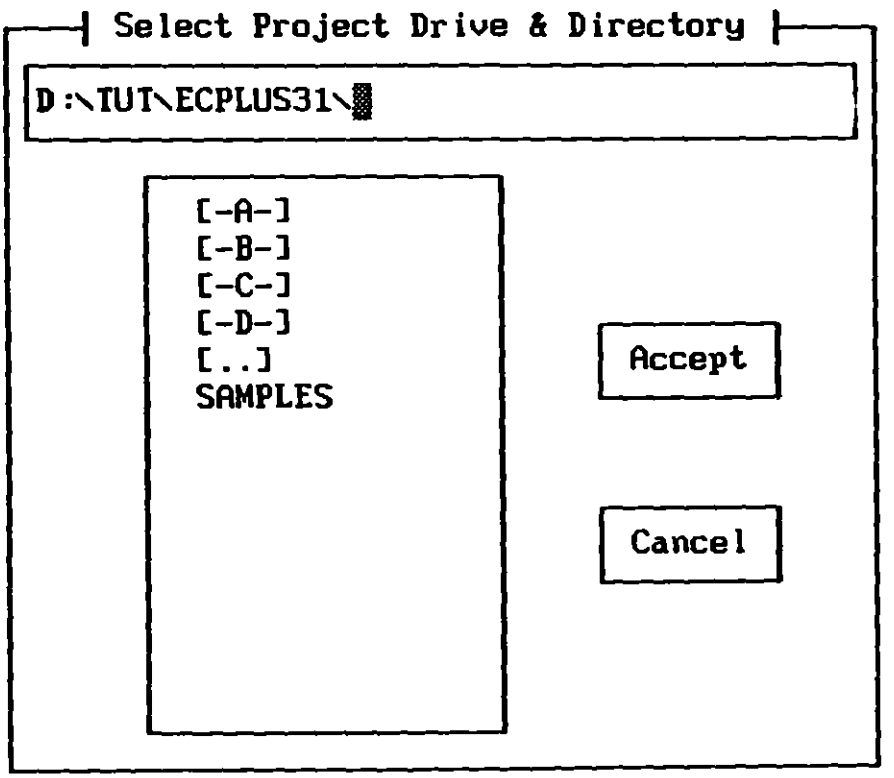


Figure 1.22 Project Directory List

To display a list of available chart files of a specific type:

- Enter a chart type after the project directory name to display a list of available charts of that type. For example:

  EASYCASE SAMPLES DFD

  displays a listing of the Data Flow Diagrams (DFD) available in the SAMPLES project directory.

To bypass the User Name Entry screen:

- Use the */N* switch. This switch tells EasyCASE Plus to use the name of the last user to log in to EasyCASE Plus. The command is:

  EASYCASE /N

  When EasyCASE Plus starts, you are not prompted for your name.

LESSON 2
CREATING A CHART

Overview

This lesson covers:

- Beginning a new chart
- Placing symbols on it
- Connecting the symbols together
- Labeling chart objects
- Adding a title to the chart
- Saving the chart

Before you start this lesson, you must have installed EasyCASE Plus with the Samples project. You should also be familiar with the material covered in Lesson 1.

EasyCASE Plus should now be running. If it is not, follow the steps provided in Lesson 1 to run EasyCASE Plus and access the sample diagram.

It should take you approximately 45 minutes to complete this lesson.

Beginning a New Chart

A project contains chart files that contain objects and data dictionary entries (storing the object definitions) in common. These chart files may also be linked in a hierarchical structure. Whenever you create (and save) a new chart, the chart files are added to the current project directory (in this case, the SAMPLES project). All of the components of a project are stored in this common project directory.

You use the 'New Chart' option in the File menu to create a new chart, of the current type (in this case, DFD).

A new, blank chart with the temporary file name of UNTITLED.DFD is displayed in the chart window.

The title block, located initially at the top left of the chart window, displays current chart information, including its type, filename, user name and date.

You will use this chart to create a new context diagram for a sample order processing system. This order processing system handles information for management, payroll, inventory, and customers. It consists of a central data process, which defines the entire processing performed by the system; several external entities, which represent things external to the system, with which the system interacts; and several connections which carry data into and out of the system, between the central data process and the external entity symbols. All symbols and connections are labeled with text to describe their function.

Adding a Symbol

The first step in creating your chart is to add symbols to it. The types of symbols available for placement on this (data flow diagram) chart are listed textually in the Objects menu, and are represented graphically in the form of icons down the left edge of the screen.

The vertical order of the object icons displayed down the left side of the screen corresponds to the order of object types listed in the Objects menu.

For the Tutorial, you will use the default chart editor settings when creating the chart.

In this lesson, you will first place all of the symbols before proceeding to label and connect them together.

NOTE: The default setting for a DFD determines that the Yourdon symbol set is used. At the end of this lesson, some of the available options are described.

First, place a Data Process symbol:

1. Select the Objects menu.

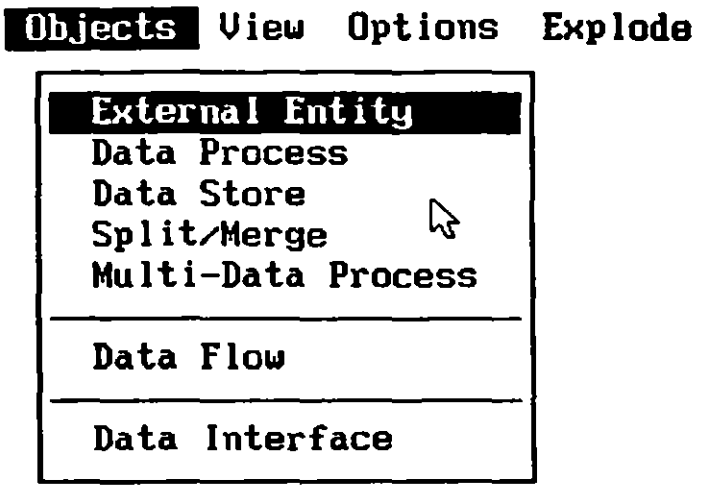


Figure 2.1 Objects Menu for a DFD

2. Select the 'Data Process' object type entry.

3. Position the mouse pointer in the center of the grid square located at column 4 row 5.

4. Click the left mouse button.

5. A circular data process symbol is displayed and appears selected (highlighted and enclosed by small boxes). This data process will represent the processing unit for the entire order processing system.

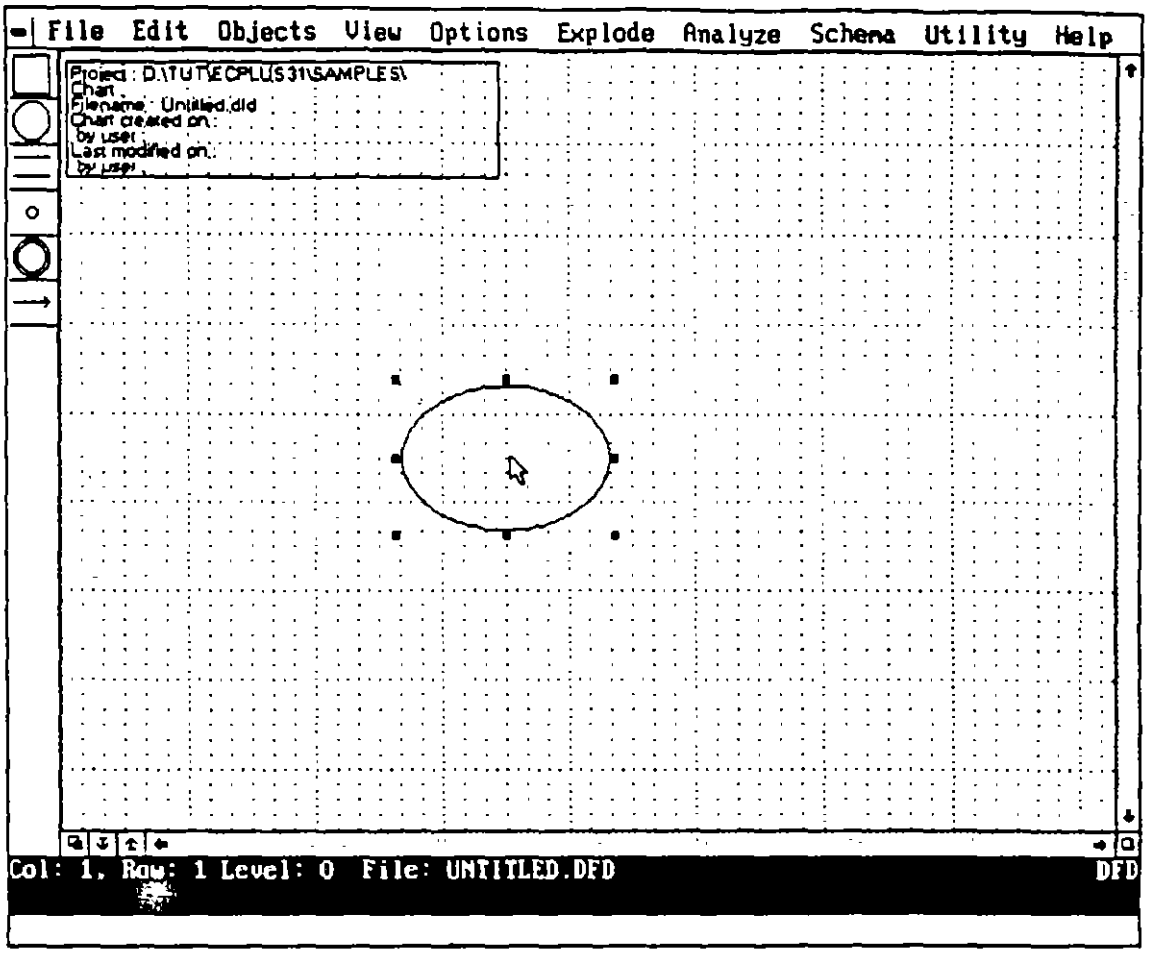


Figure 2.2 Data Process Placed and Selected

Next, place some External Entity symbols:

External entities are real world objects the order processing system interacts with.

1. Select the 'External Entity' object type entry from the Objects menu. Alternatively, select the External Entity icon (rectangle) displayed at the left of the chart window.

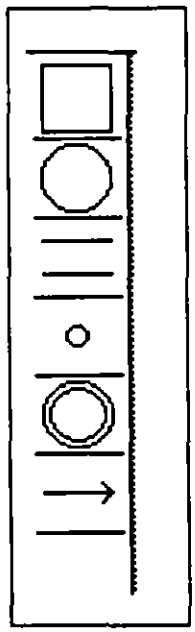


Figure 2.2a Object Icons for a Yourdon/Demarco DFD

2. Position the mouse pointer near the upper left corner of the chart window.

3. Click the left mouse button.

4. A rectangular External Entity symbol is displayed and selected.

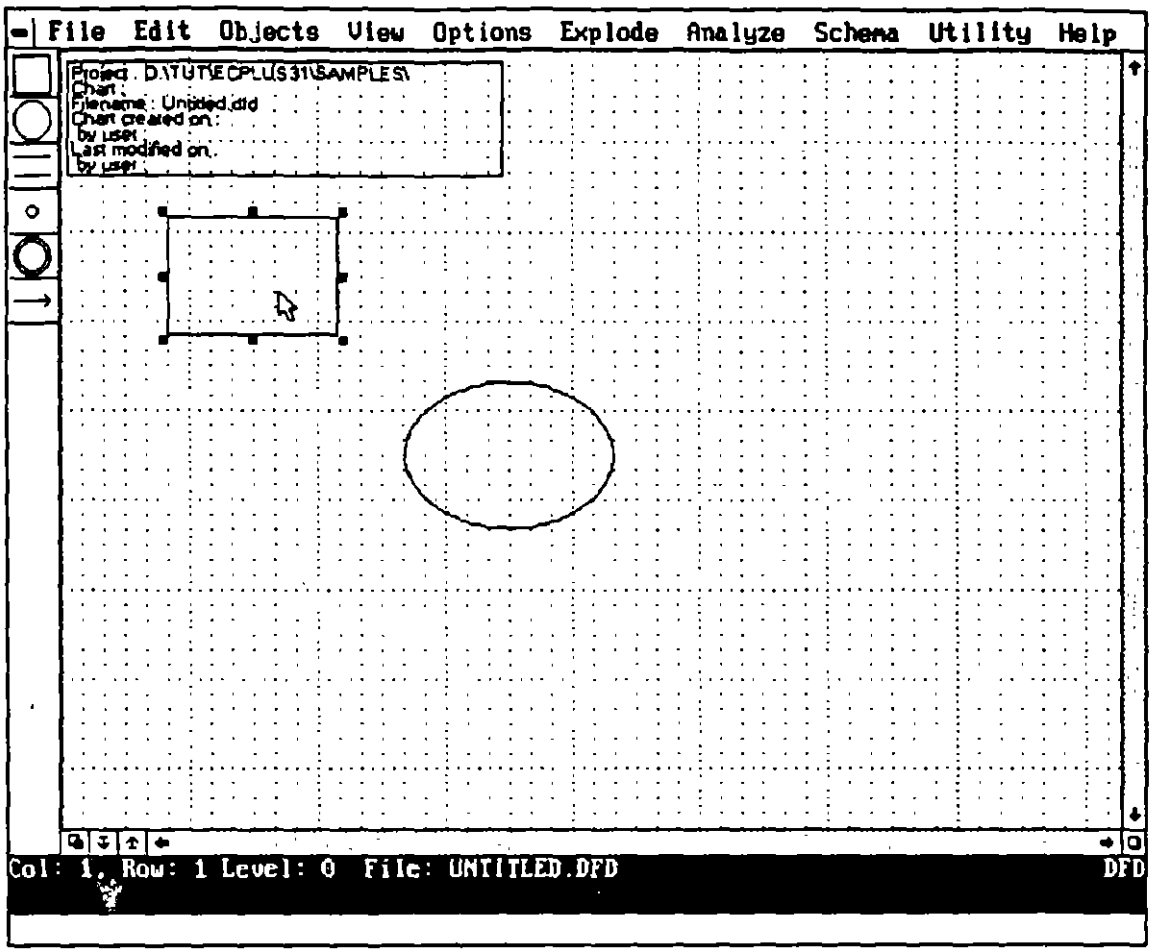


Figure 2.3 External Entity Symbol Placed and Selected

5. Place three more External Entity symbols, one each in the upper right, lower right and lower left corners of the chart window. You do not need to choose the External Entity object type again to place a symbol. Symbol placement is a repeatable command.

When you have finished, the chart window should look like this:

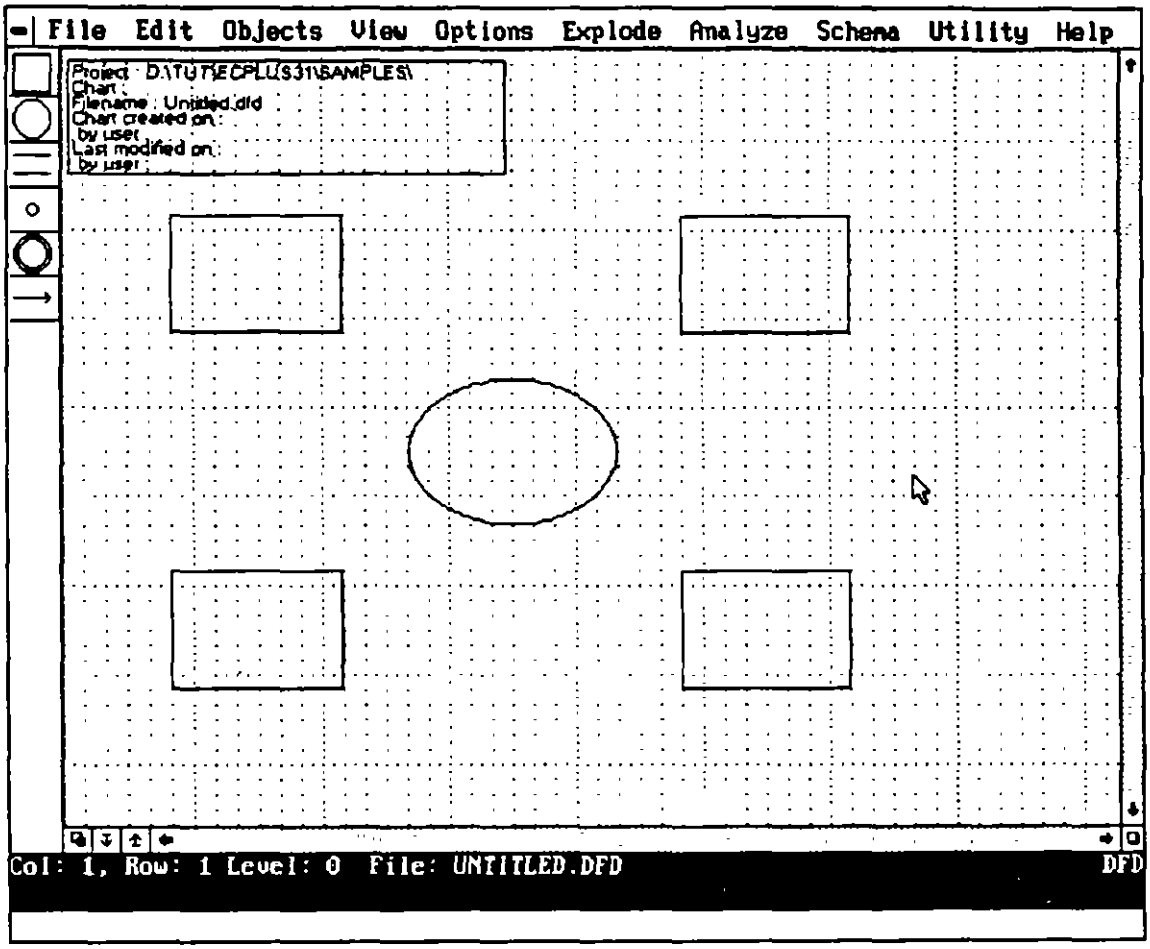


Figure 2.4 Chart with all Symbols Placed

NOTE: If you make a mistake and want to cancel an action (such as selecting an object), click the right mouse button. To delete an unwanted object, select the object and choose the 'Delete Object/Group' option from the Edit menu, or press the DEL key. You will be prompted to verify the delete.

Labeling the Symbols

Having just positioned a number of symbols on the chart, you should then label them according to their various functions. You will use the Label Object command from the Edit menu.

First, label the central Data Process symbol:

1. Select the Data Process symbol by clicking on it (be sure to click near the center of the symbol). It will become highlighted.

2. Select the 'Label Object' option from the Edit menu.

3. A text entry window is displayed in the center of the screen for you to enter the label text.

4. Type "Order Processing System". Press the ENTER key between each word to place them on separate lines as shown in Figure 2.5.

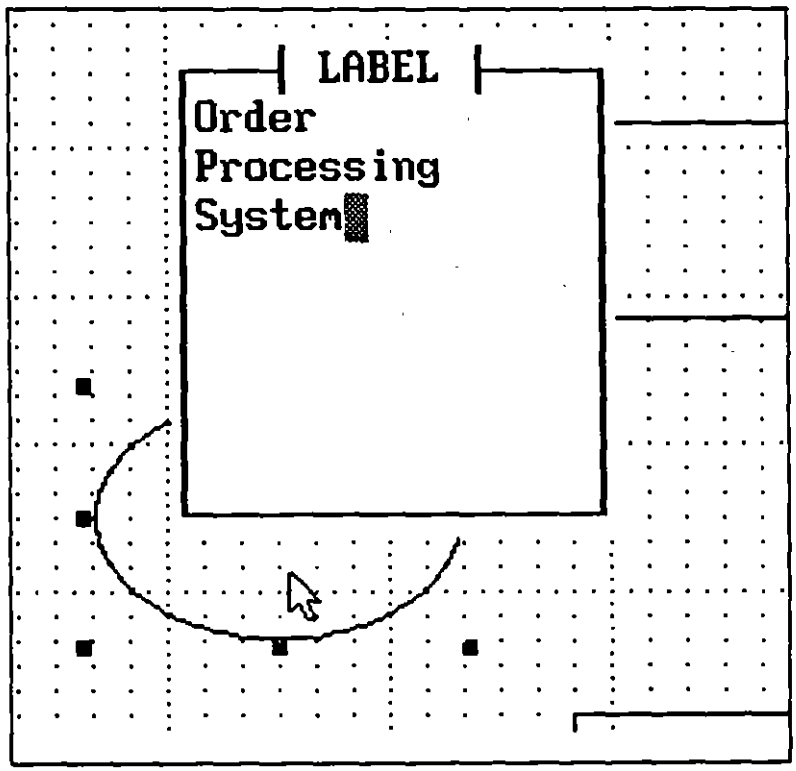


Figure 2.5 Symbol Label Text Entry Window

5. Press the F3 key when you have entered the text.

6. The label is displayed at the upper left of the symbol and selected (highlighted and enclosed within a dashed box).

7. Position the mouse pointer over the dashed box, then press and hold down the left mouse button.

   A small hand icon is displayed, positioned automatically over the label. This indicates that you can move the label by moving the mouse while you are holding down the left mouse button, as shown in Figure 2.6.

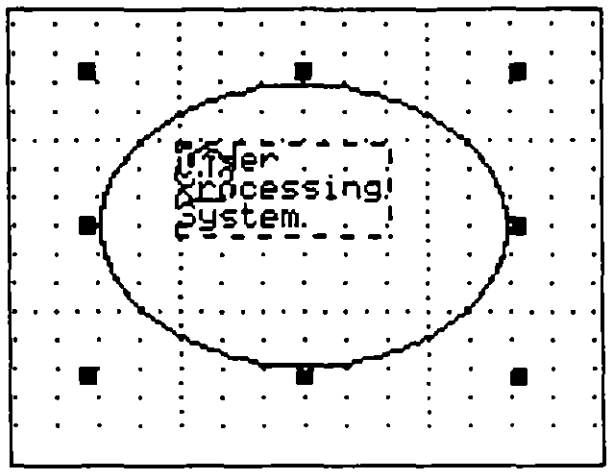


Figure 2.6 Hand Icon Positioned to Move Symbol Label

8. While still holding down the left mouse button, use the mouse to drag the dashed box to the center of the symbol as shown in Figure 2.7.

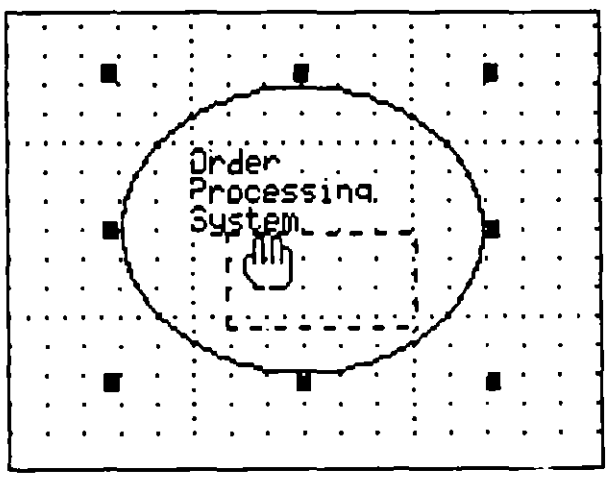


Figure 2.7 Hand Icon Moving Symbol Label

9. Release the left mouse button.

10. The text will move to that position, as shown in Figure 2.7a.

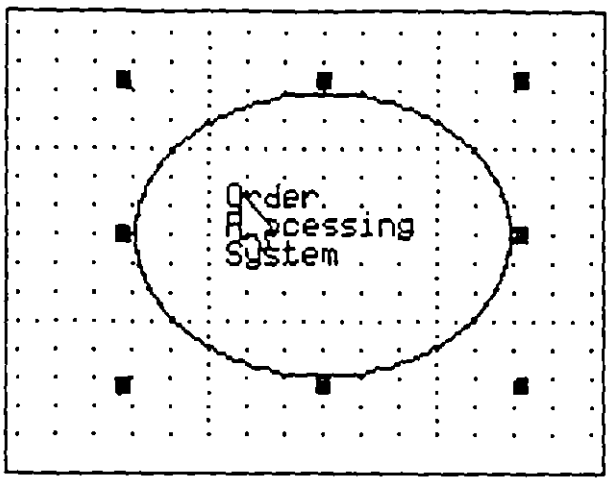


Figure 2.7a Final Symbol Label Placement

NOTE: If you want to change the text layout of your label, select the label, then choose Label Object again. You can edit the text as you wish. Remember that the label text editing operation is completed whenever you press F3.

Next, label the External Entity symbols:

1. Select the upper left External Entity symbol by clicking on it. It will be highlighted.

2. Choose the Label Object command again. Alternatively, press the ALT+L key combination.

3. Type "Customer" in the label text box, then press the F3 key.

4. Position the label in the center of the symbol, as you did for the Order Processing System data process symbol.

5. Repeat Steps 1 to 4 for the remaining three External Entity symbols labeling them as follows:

   (Upper right) "Supplier"

   (Lower right) "Payroll"

   (Lower left) "Management"

NOTE: If you wish, you can use the arrow keys on your keyboard to position the label, rather than using the mouse. Press the ENTER key when you are finished moving the dashed box to complete the operation.

When you are finished, your chart should look like this:

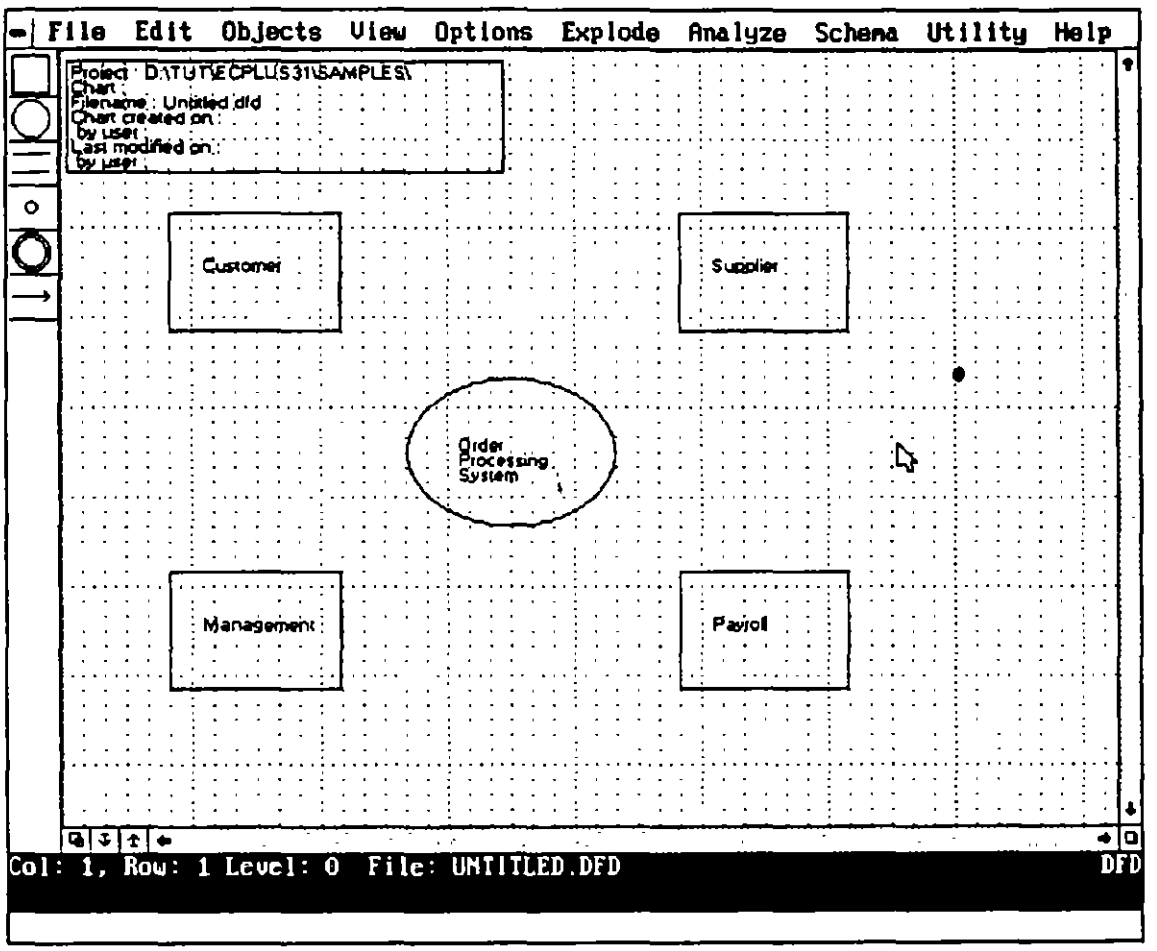


Figure 2.8 Chart with all Symbols Labelled

Connecting the Symbols

A number of symbols have been placed and labeled on the chart. The next step is to add connections between them. To do this, you will use the Data Flow option from the Objects menu, or the data flow icon.

To connect the "Order Processing System" and "Customer" symbols together:

1. Select the 'Data Flow' option from the Objects menu.

2. Select the central data process symbol labeled "Order Processing System" by clicking on it (click near the center).

   This will be the source symbol for the data flow. A number of small boxes are displayed around the symbol, as shown in Figure 2.9. These represent discrete positions at which a connection can exit the symbol.

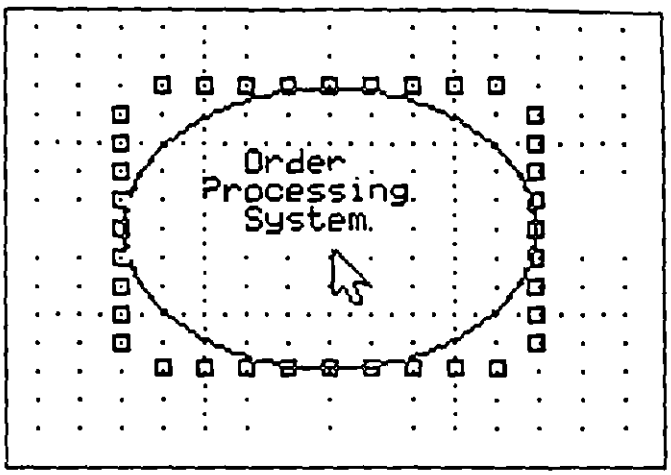


Figure 2.9 Selected Source Symbol for Data Flow to Begin

3. Select the box half way down the left side of the symbol by clicking on it using the mouse.

   The small box that you selected remains displayed at this location, and the other boxes are erased (see Figure 2.10). This indicates where the connection will leave the symbol.

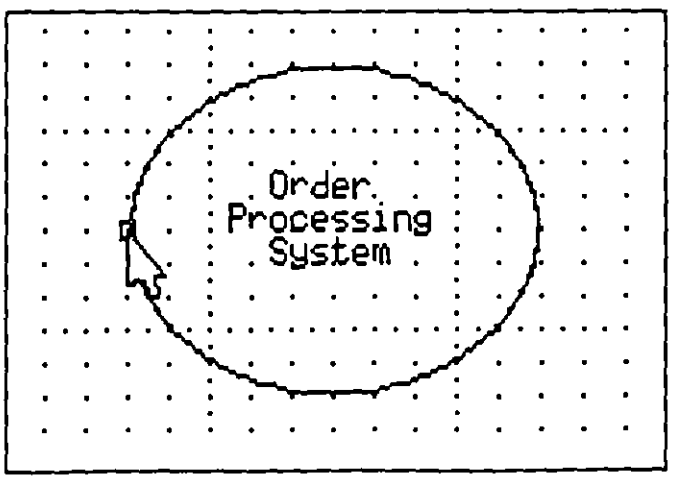


Figure 2.10 Selected Origin for Connection Attachment

4. Select the External Entity symbol labeled "Customer" by clicking on it.

   This will be the target symbol for the data flow. A number of small boxes are displayed around this symbol, as shown in Figure 2.11. These represent discrete positions at which a connection can enter the symbol.

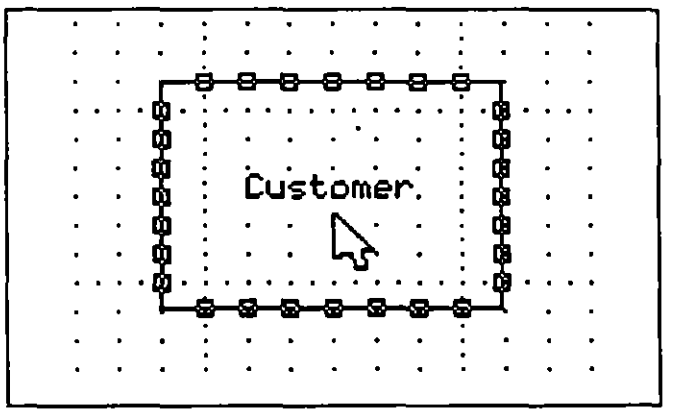


Figure 2.11 Flow Entry/Exit Ports

5. Select the box midway on the lower side of the Customer symbol by clicking on it, as shown in Figure 2.11a.

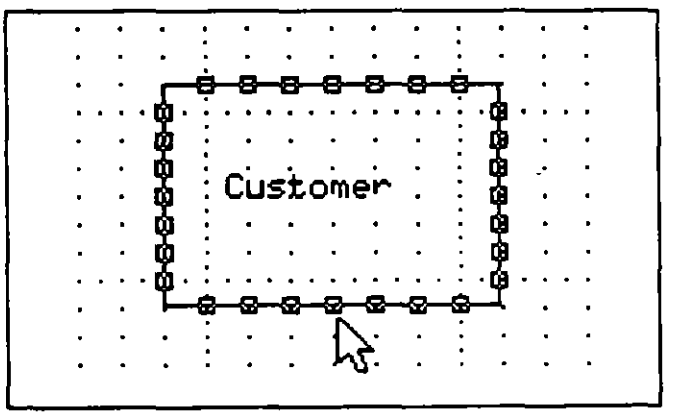


Figure 2.11a Select Flow Entry Port for Customer Symbol

A connection is then drawn automatically from the "Order Processing System" symbol to the "Customer" symbol, using a horizontal and a vertical line segment, and with an arrowhead at the destination end, as shown in Figure 2.12.

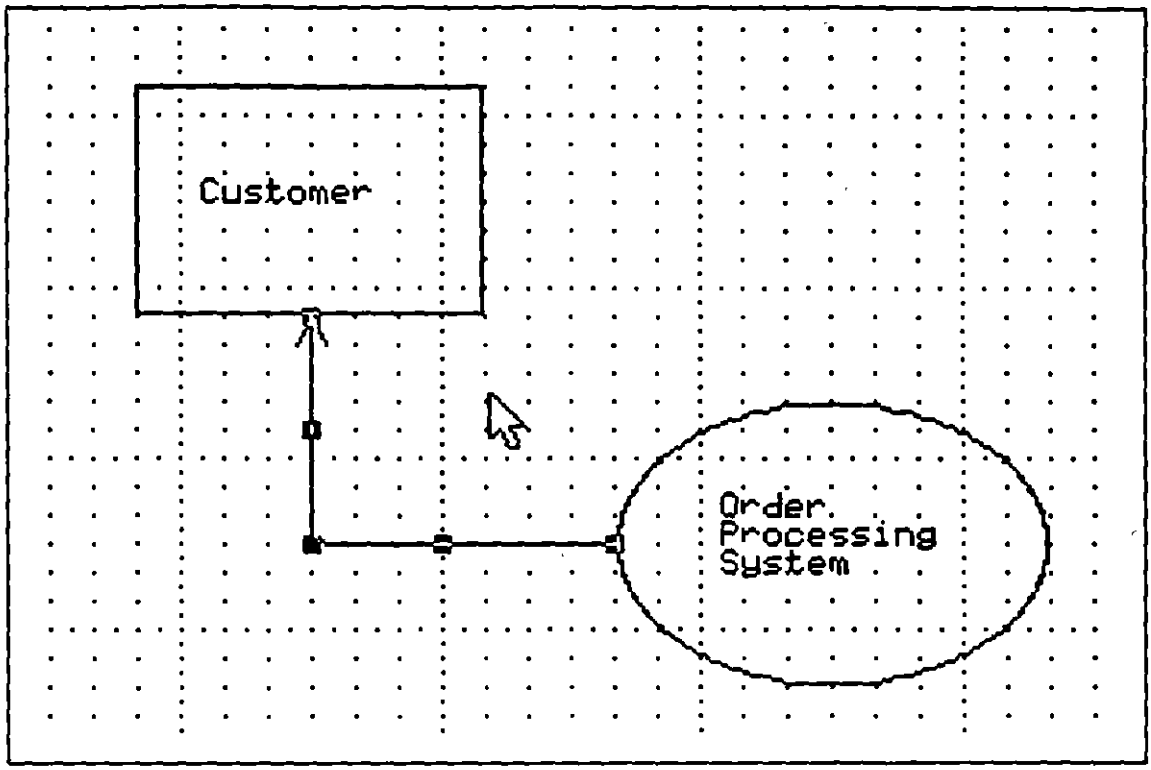


Figure 2.12 Initial Connection Placed

6. Repeat Steps 1 to 5 using the "Customer" external entity as the source symbol and the "Order Processing System" data process as the target symbol.

   Route the connection from the right side of the "Customer" symbol to the top of the "Order Processing System" symbol, as shown in Figure 2.13.

The completed data flow connections are displayed, connecting the two symbols at the points you have chosen. There is a data flow *from* the Order Processing System *to* the Customer, and a different data flow back to the Order Processing System.

Your chart should now look like this:

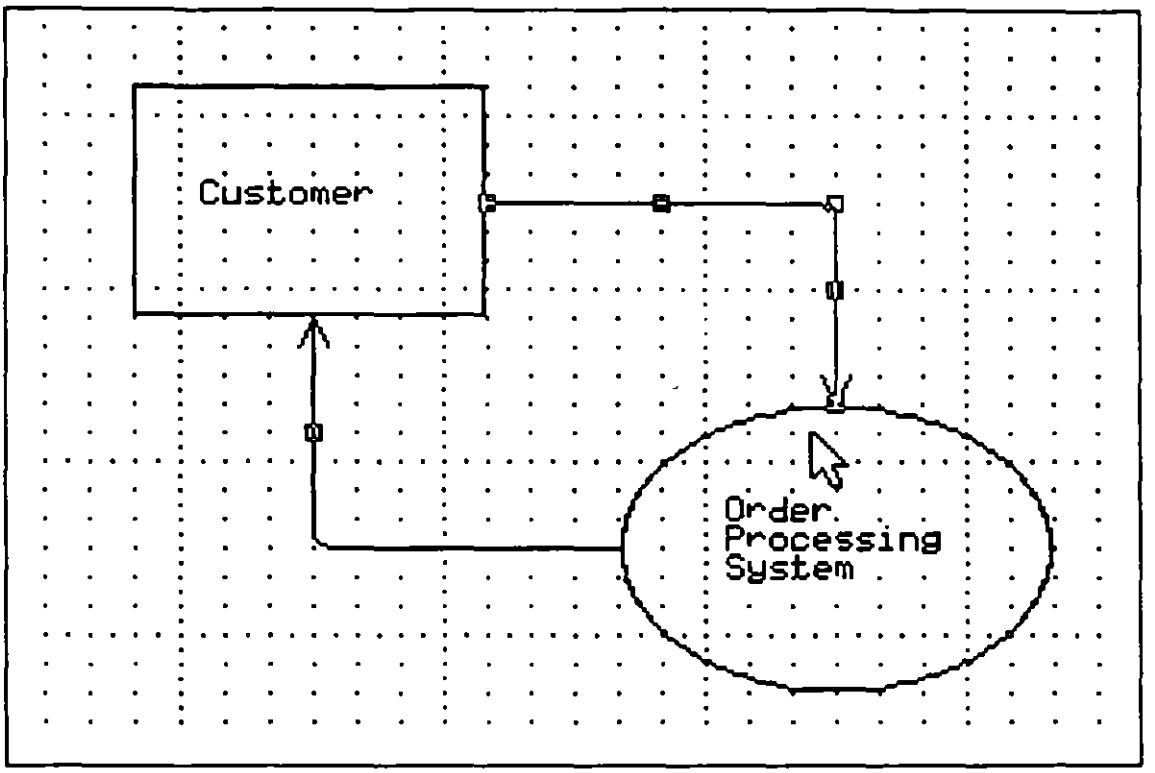


Figure 2.13 Data Flows Connecting a Data Process and an External Entity

Next, place connections for the remaining symbols. Refer to the chart in Figure 2.14 to complete your connections. Refer to the following table which shows the source and target symbols for the data flow connections:

Source Symbol	Target Symbol
Customer*	Order Processing System*
Order Processing*	Customer*
Order Processing System	Supplier
Supplier	Order Processing System
Order Processing System	Payroll
Payroll	Order Processing System
Order Processing System	Management
Management	Order Processing System

Table 2.1 Source and Target Symbols for Connections

*Connections already placed in preceding sections.

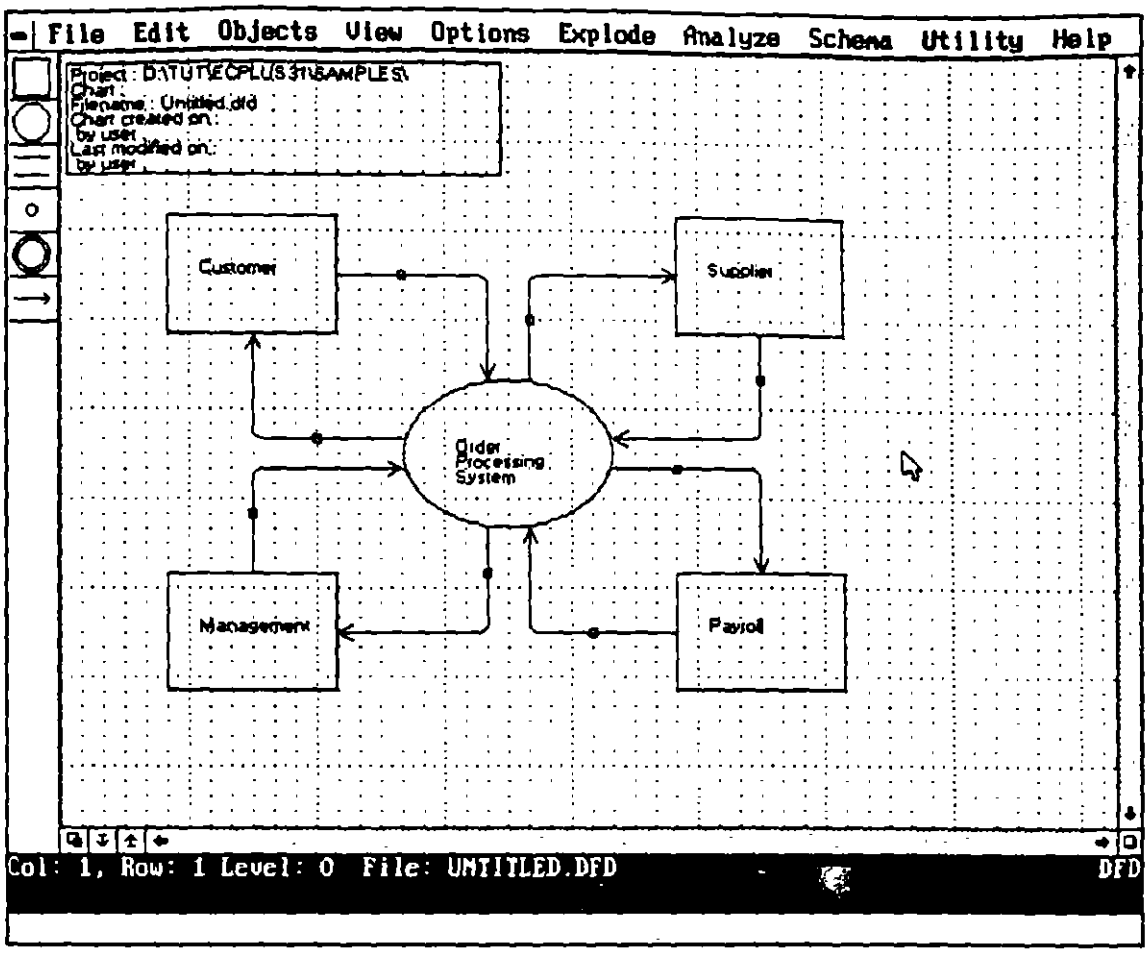


Figure 2.14 Chart with all Connections Completed

Adding Labels to the Connections

So far, your chart has all of the symbols labeled and connections placed. The next task is to label the connections between the symbols. This procedure is similar to labeling symbols (see the previous section, "Labeling a Symbol").

To label a connection:

1. Select the connection flowing from the "Order Processing System" data process symbol to the "Customer" external entity symbol, by clicking on the small box ('handle') on the connection. It will be highlighted and small boxes will be located at each end, and also at the corner and midpoint of each line segment.

2. Choose the Label Object command from the Edit menu, or press ALT+L.

3. Type "Statement" in the label text entry box that appears, as shown in Figure 2.15.

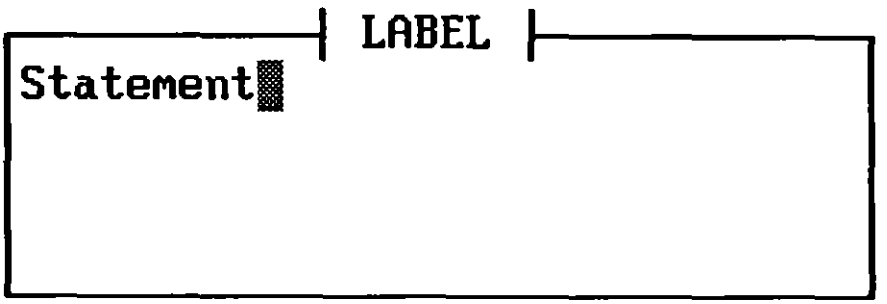


Figure 2.15 Connection Label Text Entry Box

4. Press the F3 key.

5. The label is displayed inside a highlighted, dashed box next to the connection's 'handle'.

6. Using the mouse, position the connection label, as you did for the symbol labels, by dragging it so that it is located next to the connection, as shown in Figure 2.16.

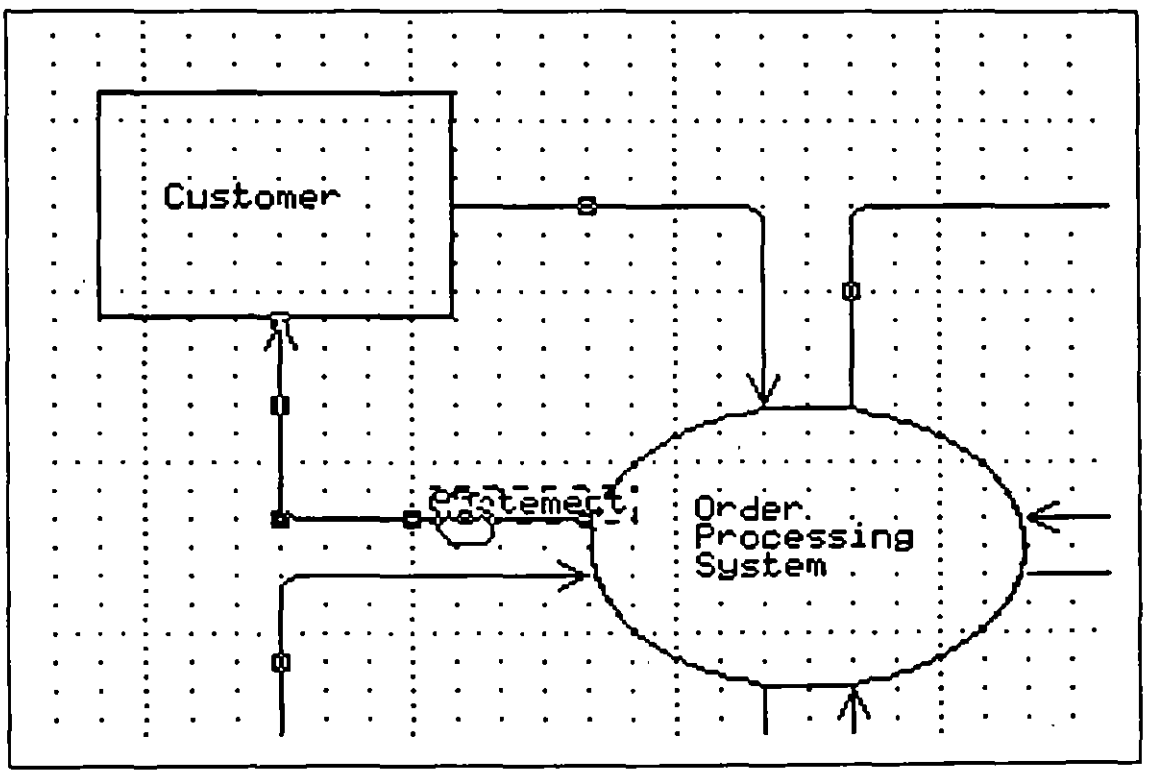


Figure 2.16 Hand Icon Positioned for Moving Connection Label

Repeat steps 1 to 6 above for each other connection on the chart, using Table 2.2 as a guide to the label to use for each remaining connection.

Refer to the following table and Figure 2.17 to label your connections.

Source Symbol	**Target Symbol**	**Connecting Label (used in next step)**
Order Processing System	Customer	Statement*
Customer	Order Processing System	Payment
Order Processing System	Supplier	Inventory Request
Supplier	Order Processing System	Inventory Delivery Report
Order Processing System	Payroll	Sales Commission Request Form
Payroll	Order Processing System	Sales Commission
Order Processing System	Management	Sales Report System
Management	Order Processing System	Sales Quotas

Table 2.2 Label Text for Connections

*Connection already labeled in preceding section.

Upon completion, the chart should look like the following:

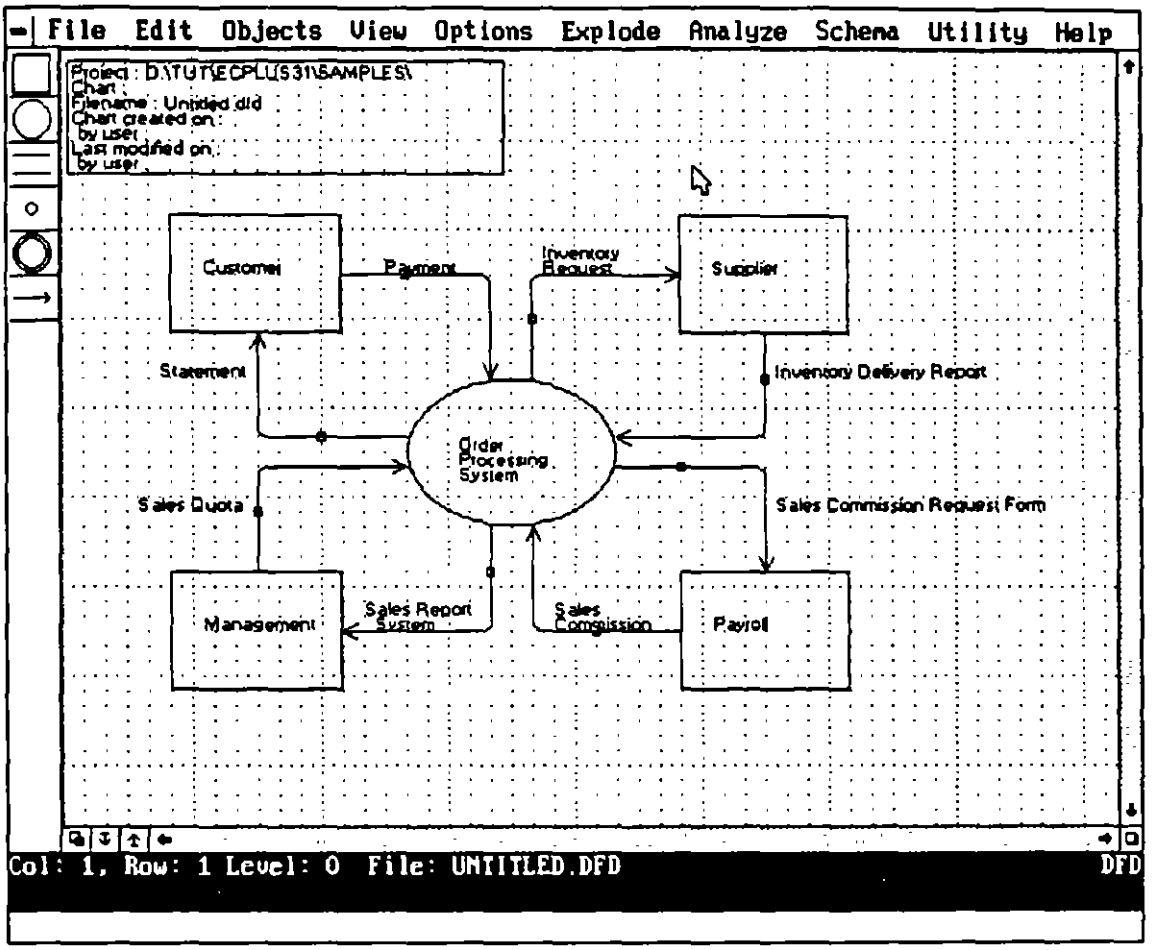


Figure 2.17 Chart with all Connections Labeled

NOTE: As an alternative to using the 'Label' option from the Edit menu (or press ALT+L) to label an object after each is placed, you can automatically invoke the Label text entry box.

To do this:

1. Choose the Options menu.

2. Select the 'Preferences' option.

3. In the dialog box that appears, click on the 'Auto Label' entry so that an 'X' appears to the left of it, as shown in Figure 2.18.

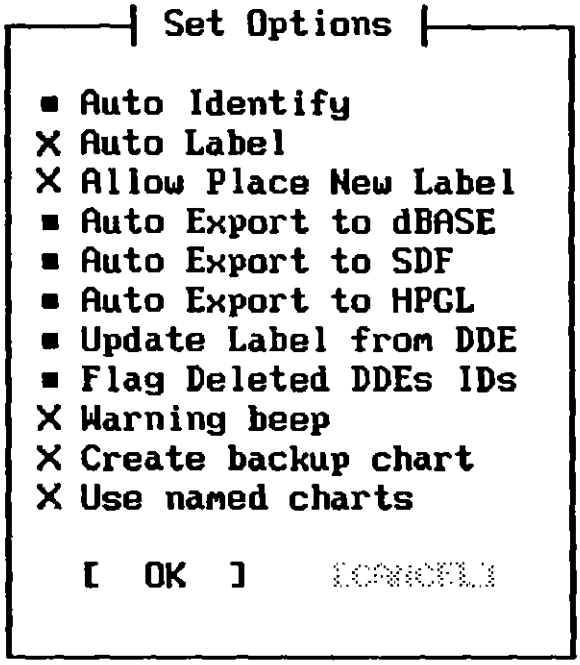


Figure 2.18 Preferences Dialog Box Containing the Auto Label Option set to On

4. Click on 'Cancel'.

Adding a Title to the Chart

Your chart should now be displayed with all symbols and connections labeled. You can add a title to describe the purpose of the chart if you like. To do this, use the 'Text Block' option from the Edit menu.

To add a title to the chart:

1. Select the 'Text Block' option from the Edit menu, or press the ALT+T key combination.

2. Position the mouse pointer in the upper right corner of the chart window, as shown in Figure 2.19.

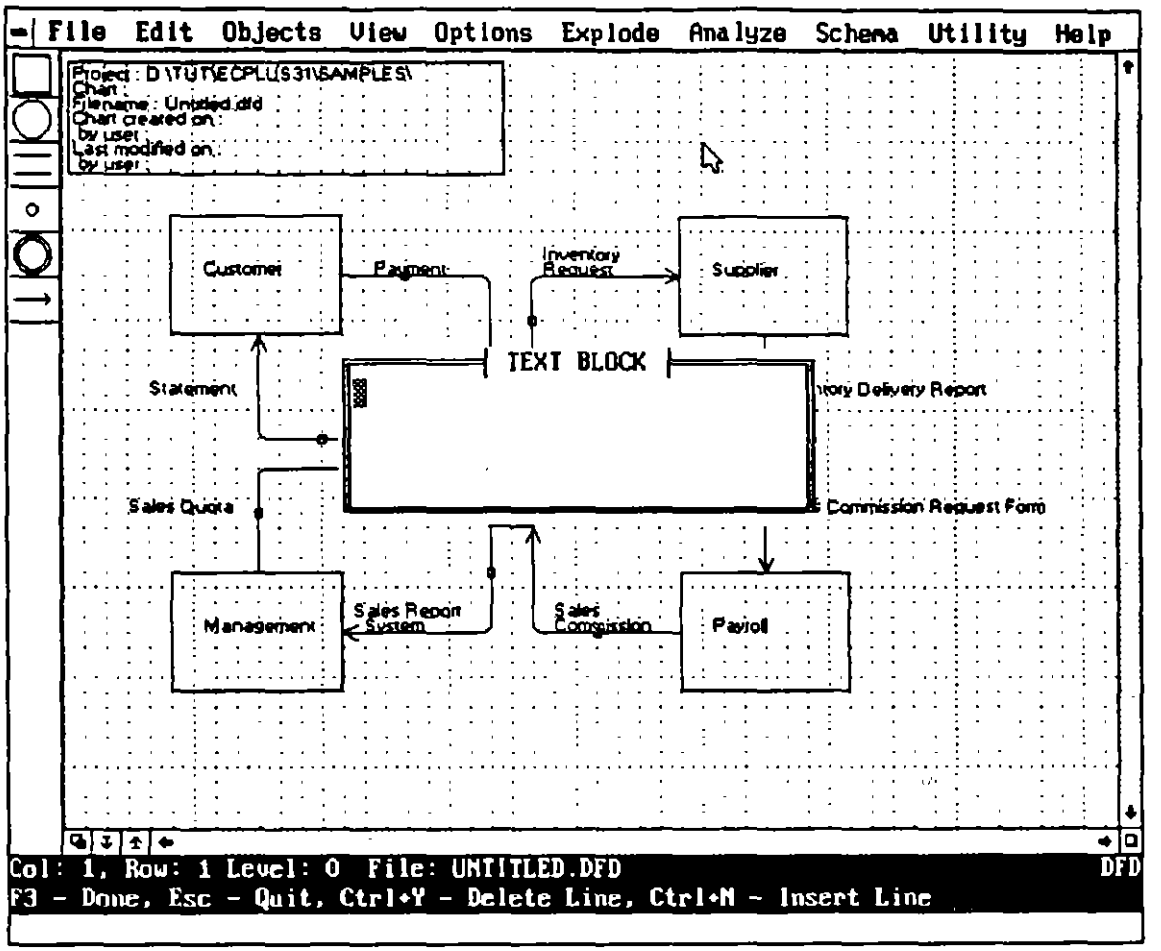


Figure 2.19 Mouse Pointer Positioned to Place Text Block

3. Click the left mouse button. This defines the location for the chart title text block.

4. A text entry box appears in the center of the screen, as shown in Figure 2.19.

5. Type "Sample Order Processing System" then press ENTER. Then type "Data Flow Diagram (DFD)" as the title for your chart, as shown in Figure 2.19a.

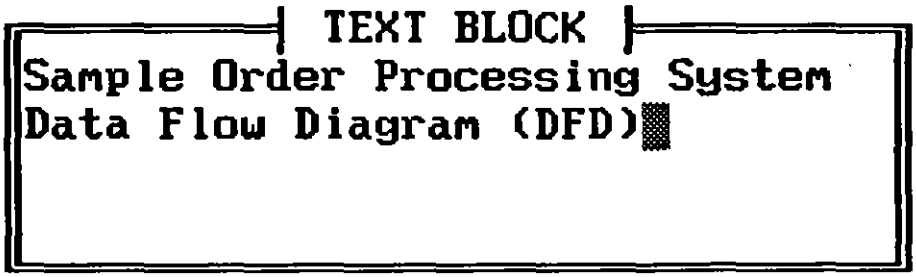


Figure 2.19a Text Block Entry Box

6. Press the F3 key to accept the title text.

7. The title is displayed at the location you chose previously (steps 2 and 3), as shown in Figure 2.20.

8. If you want to move the title, use the same procedures you used to position a label. That is, press the left mouse button, drag the text block outline to the desired location, then release the mouse button to place the text block.

9. Your completed chart should look like this:

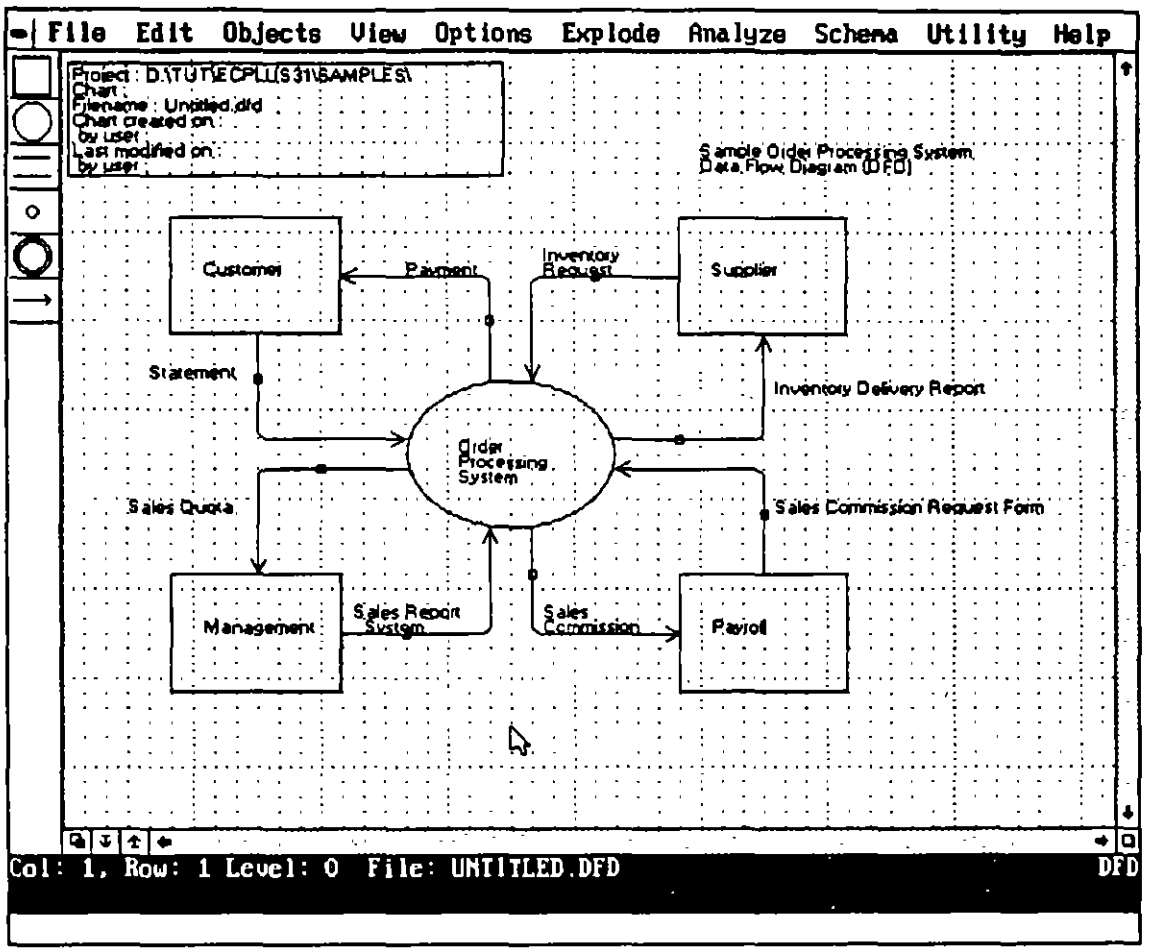


Figure 2.20 Completed Chart.

NOTE: If you want to change the text of your title, select the title then choose the 'Text Block' option again. You can edit the text as you wish.

Saving the Chart

It would be a good idea to save the chart at this point. Use the Save As command from the File menu to do this using a new chart file name.

1. Select 'Save As' from the File menu.

   You are prompted for the new filename to use.

2. Press the 'Delete' key to erase the current name (untitled.dfd).

3. Type in "tutor1". You do not need to enter a file extension, as it will default to the current chart type (DFD in the example), as shown in Figure 2.21a.

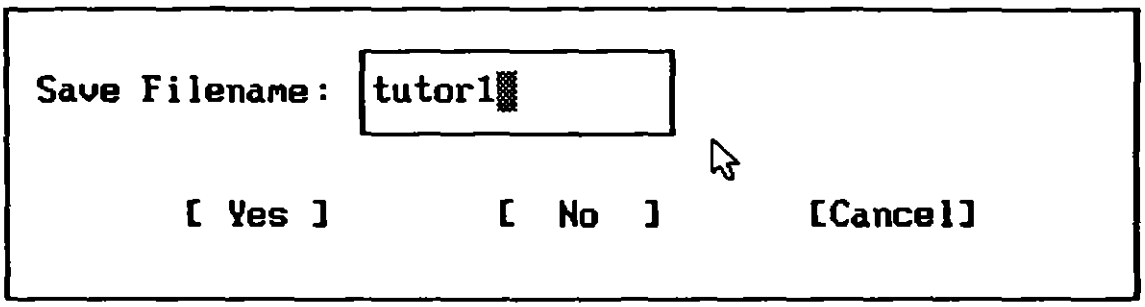


Figure 2.21a Filename Text Entry Box

4. Click on 'Yes.'

5. A listing of the current DFD chart file names is then displayed (see Figure 2.21b) and you are prompted for the chart name to use.

   The name of a new chart defaults to its file name. The chart name can be a longer (up to 32 characters) descriptive name for the chart and is entered into the data dictionary for later access.

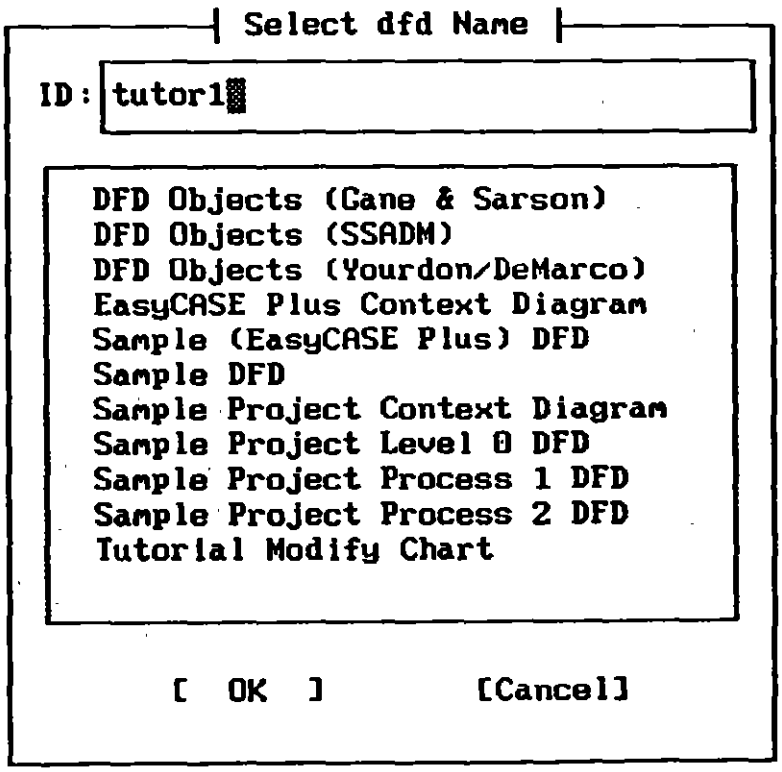


Figure 2.21b Listing of Available Chart Names with the Current Name Shown in the Upper Text Entry Box

6. Click on 'OK' to accept "tutor1" as the DFD chart name.

The chart file is saved in the SAMPLES project directory and the chart window is re-displayed. Notice that the Title Block information and status line is updated to reflect the new chart name and file name (tutor1 and tutor1.dfd, respectively). You can now continue working on the chart you have created.

Alternatives

There are a number of options you can use to customize your chart creation process. The following are some options available in the Options and View menus that relate to this lesson:

Default Symbol Set

The default symbol set used for DFDs is the Yourdon/DeMarco set. For Data Flow Diagrams, your other options are the Gane & Sarson and SSADM symbol sets. When you choose one of these options, the appearance of the chart symbols changes. Try it by choosing the 'Change Chart Type' entry in the File menu, then click on the Gane & Sarson DFD option. The chart is redrawn using this symbol set as shown in Figure 2.22. Repeat the process to choose the SSADM symbol set, and then return to the Yourdon/DeMarco symbol set.

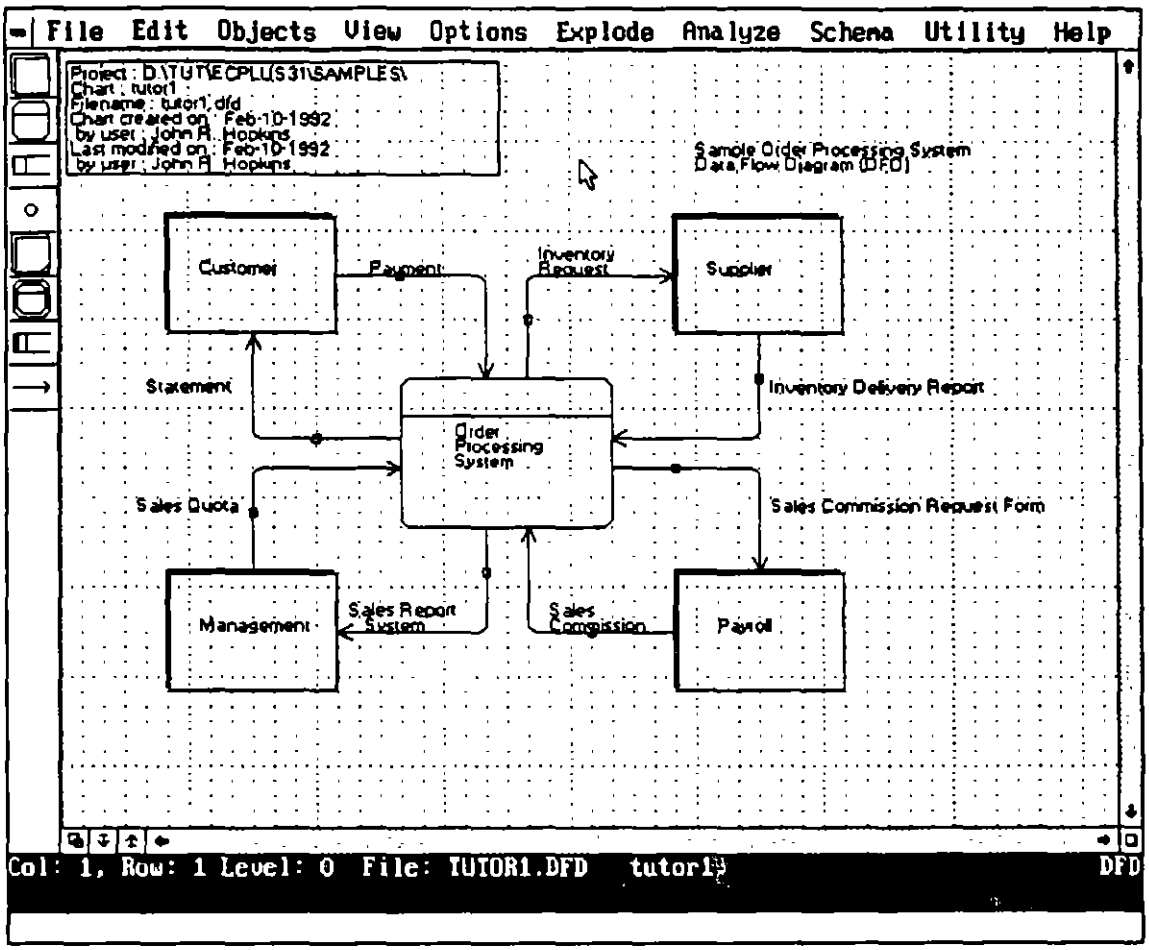


Figure 2.22 Gane & Sarson DFD Symbol Set

Default Symbol Size

The default symbol size is Normal, the size you have been using throughout this lesson. You can change this setting to Small, Large, Extra Large and Giant. If you do, each symbol you place subsequently will be drawn using the chosen size. This feature is accessed via the Default Symbol Size entry in the Options menu.

Auto Label

This setting determines whether or not you are automatically prompted to label an object when you create it. It is 'Off' by default, allowing you to create all of the objects before labelling them. When it is 'On,' the label text entry box will pop up automatically whenever you place a new symbol on the chart. This entry is found by choosing the Preferences entry from the Options menu.

Allow Place New Label

This option allows you to reposition a label after it has been placed using the mouse or keyboard. The default setting is 'On.' When this setting is 'Off,' the label is placed inside a symbol, or adjacent to the selection marker for a connection, and you cannot adjust its position. To do so, you must use 'Move Object/Group' command after placing it. This option is found by choosing the Preferences entry from the Options menu.

Automatic Flow Routing

When you place connections, this option automatically chooses a routing between symbols that accounts for the proximity of diagram elements. This option is 'On' by default. When it is 'On,' only horizontal and vertical line segments are drawn between points on connections. When you turn it 'Off,' single, straight-line segments are drawn directly between points and connections. This option is found in the Options menu.

Default # of Arrowheads

All of the data flow connections you used were one-way single-headed connections. This is the default setting. However, you can also use one-way double-headed, two-way single headed, two-way double headed, or no arrowheads on connections, by default. This feature is accessed via the Options menu.

Open Arrowheads

This setting is 'On' by default. When you turn it 'Off,' the connection arrowheads are closed in, but unfilled. This option applies to all connections on the chart. This is an option in the View menu.

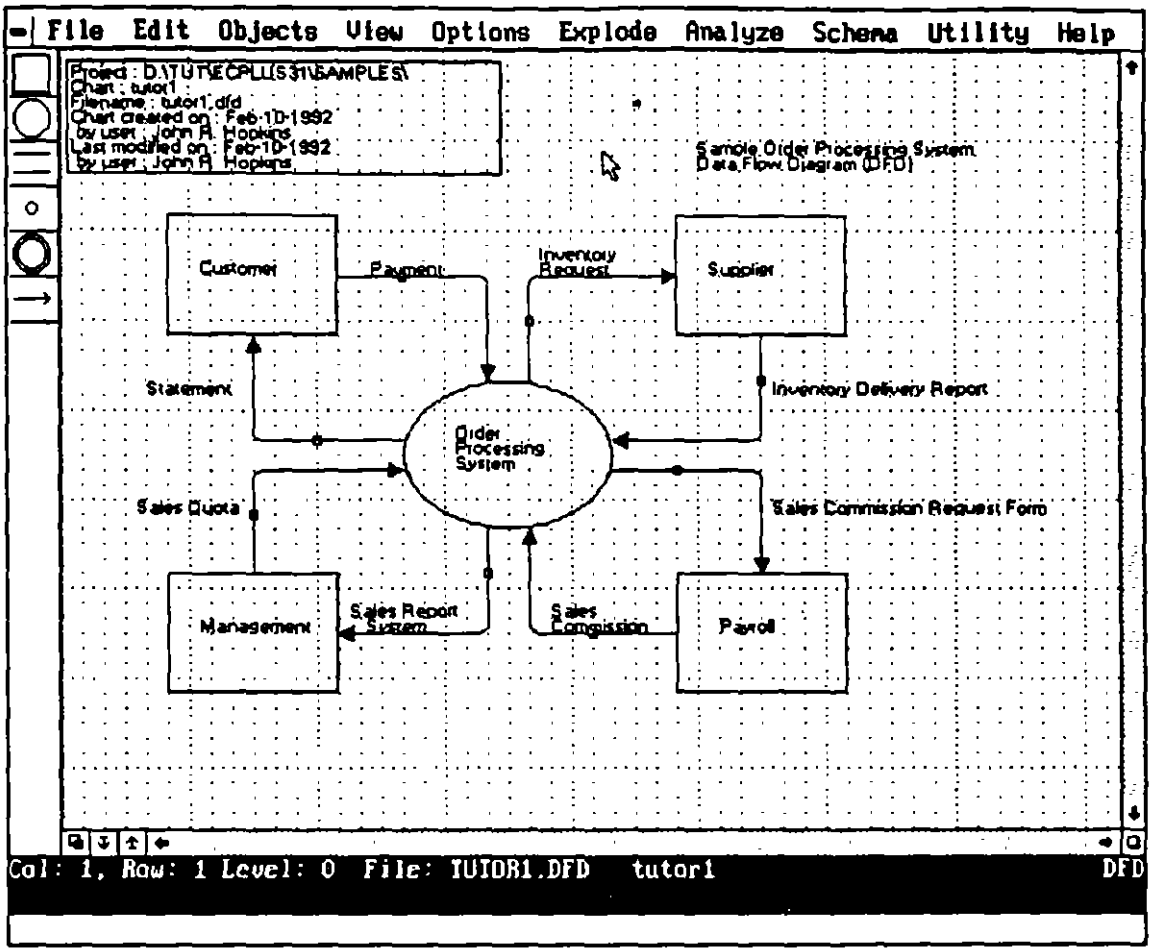


Figure 2.23 Diagram with Closed and Filled Arrowheads

Filled Arrowheads

This setting is 'Off' by default. When turned 'On,' this setting displays solid (filled in) connection arrowheads. This option applies to all connections on the chart. This is an option in the View menu.

Rounded Corners

The corners on the connections you used were rounded, the default setting. Turning this setting 'Off' produces sharp corners. This option applies to all connections on the chart. It is an option in the View menu.

Display Title Block

Turn this setting 'Off' to suppress the display of the Title Block in the upper left corner of the chart window. It is 'On' by default. This is an option in the View menu.

NOTE: The Title Block cannot be edited or deleted, however, it can be moved.

Drawing Grid

This option turns the drawing grid 'Off' for a clear chart background as shown in Figure 2.24. It is 'On' by default. Even when turned 'Off,' the grid is always active. This is an option in the View menu.

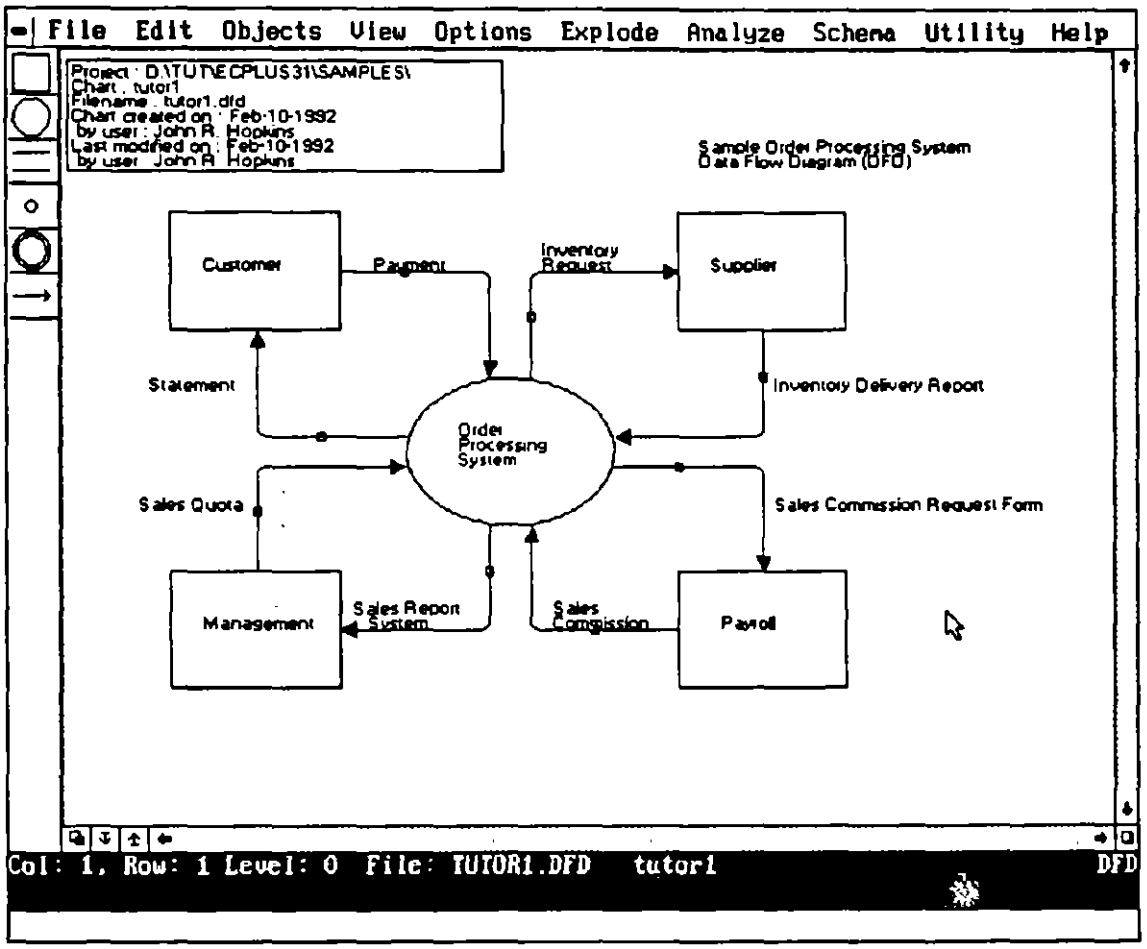


Figure 2.24 Chart with Drawing Grid Off

Auto Redraw After Edit

This option is found in the Options menu and defaults to 'Off.' When it is 'On,' then wherever you use an editing command (such as Move or Delete Object/Group), the screen will be redrawn automatically on completion of the edit operation. This removes the need for you to click on the redraw icon (or select Redraw chart from the View menu, or press CTRL + R) to redraw the screen. Screen redraws may be necessary after an editing operation to remove any screen 'debris' resulting from the edit.

LESSON 3
MODIFYING A CHART

Overview

For this lesson, you will start with a chart that is incomplete, contains some deliberate errors, and is in need of repair. In fixing and completing the chart, you will use the following functions:

- Loading a chart
- Moving a symbol
- Copying a symbol
- Deleting a symbol
- Resizing a symbol
- Moving a block of symbols
- Copying a block of symbols
- Deleting a block of symbols
- Changing a symbol label
- Changing a symbol type
- Changing connections
- Moving the entire chart

To complete this lesson, you will need to be familiar with the material covered in Lessons 1 and 2.

This lesson should take you approximately 45 minutes to complete.

NOTE: You should be using the default settings for EasyCASE Plus. If you have changed any settings, restore the default values for the duration of this lesson. Otherwise, the results of the instructions may be different.

Loading a Chart

First you will need to load the chart to modify. Use the MODIFY.DFD chart stored in the SAMPLES project directory. This chart is a variation on the TUTOR1.DFD chart you created in Lesson 2.

To load the chart:

1. Choose the Load Chart command from the File menu, or press CTRL + L.

   A dialog box is then displayed listing the data flow diagram (DFD) charts in the SAMPLES project directory that are available for loading.

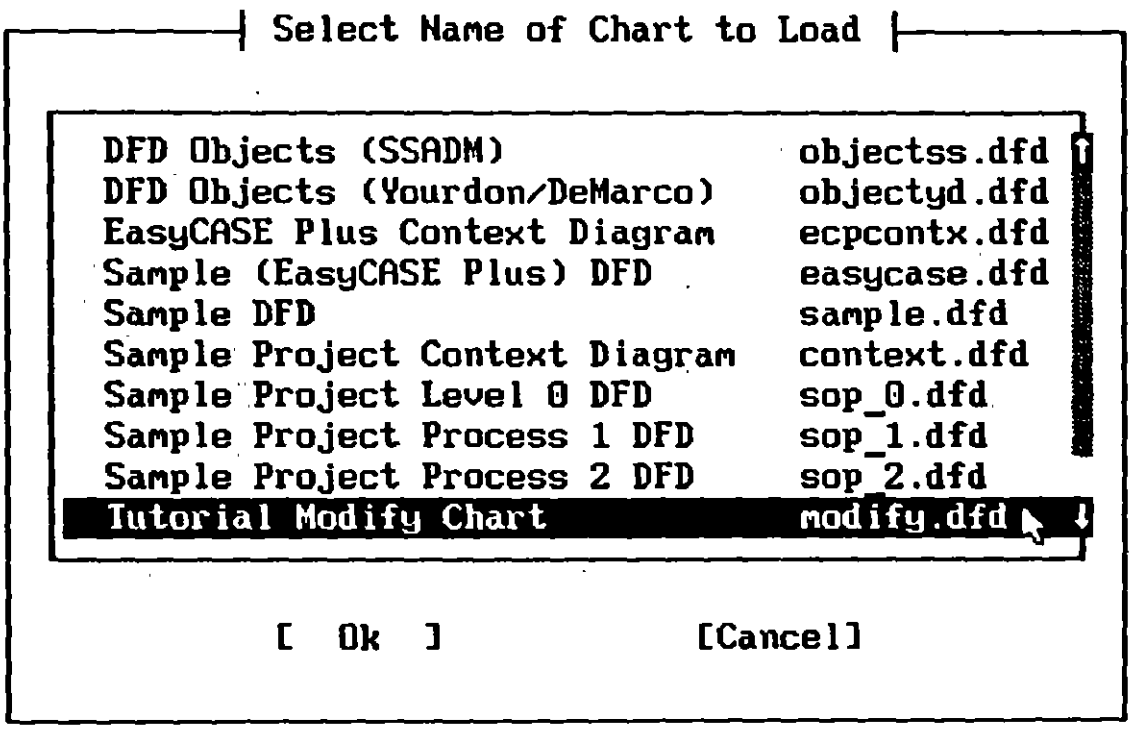


Figure 3.1 Listing of Available DFD Chart Files

2. Select the MODIFY chart entry (MODIFY.DFD) by clicking on it.

3. Click on 'OK.'

   Alternatively, double click on the chart entry.

4. The chart is then loaded and displayed in the chart window, as shown in Figure 3.2.

You will observe that there are a number of problems with the layout of this diagram, which you are about to correct in the following steps. These 'corrections' will allow you to gain familiarity with the methods available for manipulating existing chart objects.

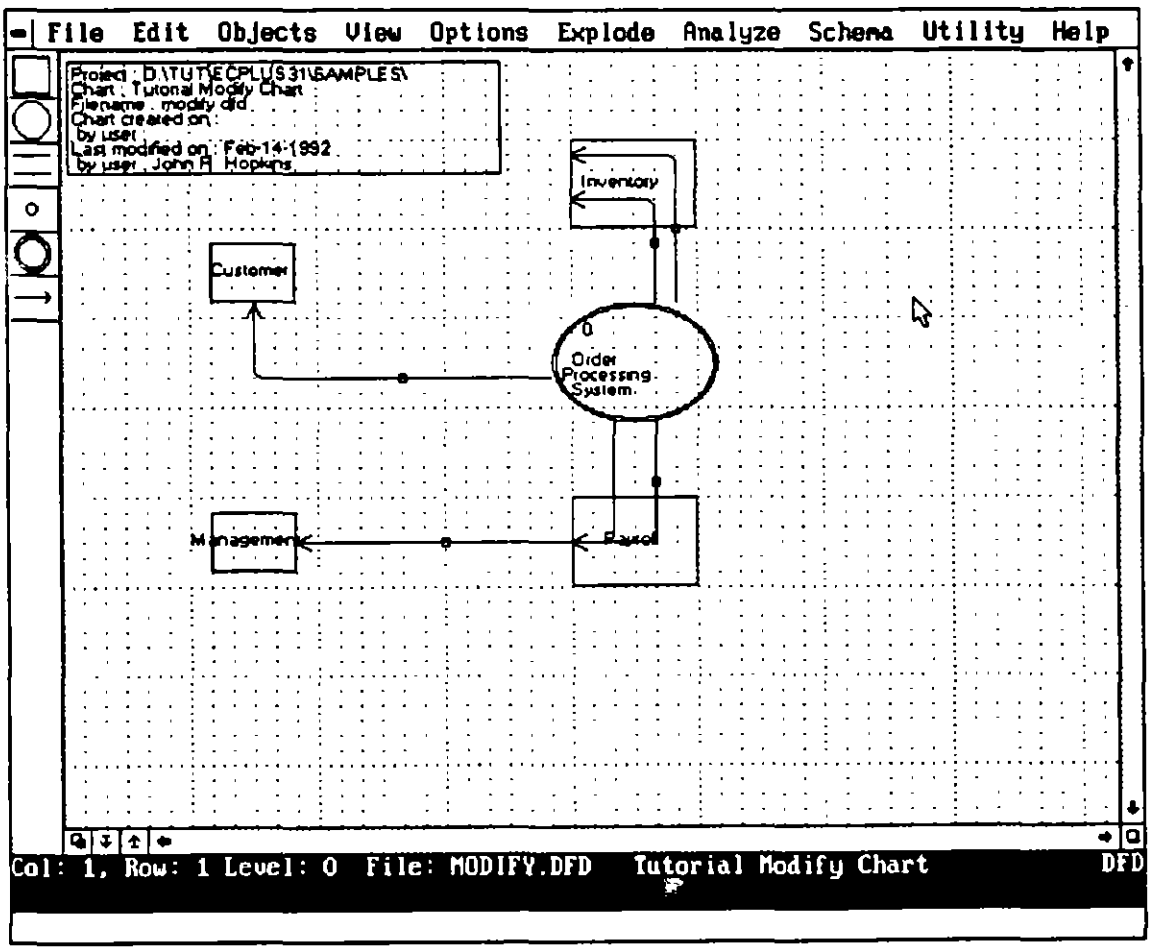


Figure 3.2 Chart to be Worked On in this Lesson

Moving a Symbol

Among the problems you will notice on the chart is the placement of the symbols. You will begin modifying the chart by moving the Order Processing System data process symbol to a more central location.

To move the data process symbol labeled "Order Processing System":

1. Position the mouse pointer over the data process symbol labeled "Order Processing System" (be sure to locate it in the center of the symbol).

2. Press and hold down the left mouse button.

   The symbol is highlighted and a small hand icon is displayed (see Figure 3.3). This indicates that the symbol is selected and you can move it by moving the mouse while you are holding down the left mouse button.

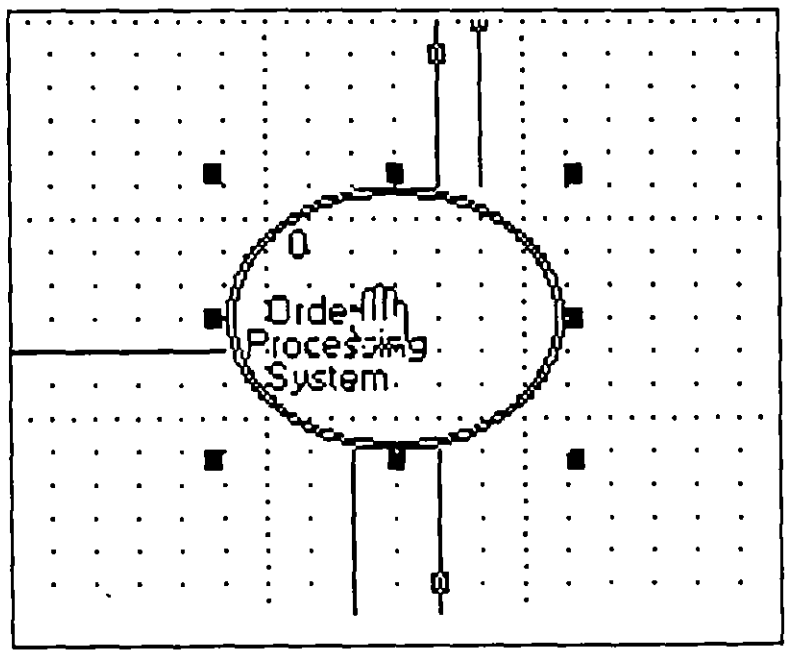


Figure 3.3 Hand Icon Positioned Ready to Move Symbol

3. While still holding down the left mouse button, move the mouse to move the hand icon toward a central location between the four surrounding (external entity) symbols. An outline of the symbol follows the mouse cursor.

4. Release the left mouse button.

5. The symbol will move to the new location. The connections and their associated labels are automatically redrawn for the new position, as shown in Figure 3.4.

6. Click on the Redraw icon at the bottom right corner of the screen to refresh the screen and remove any 'debris' resulting from the move.

When you have finished, your chart should look like this:

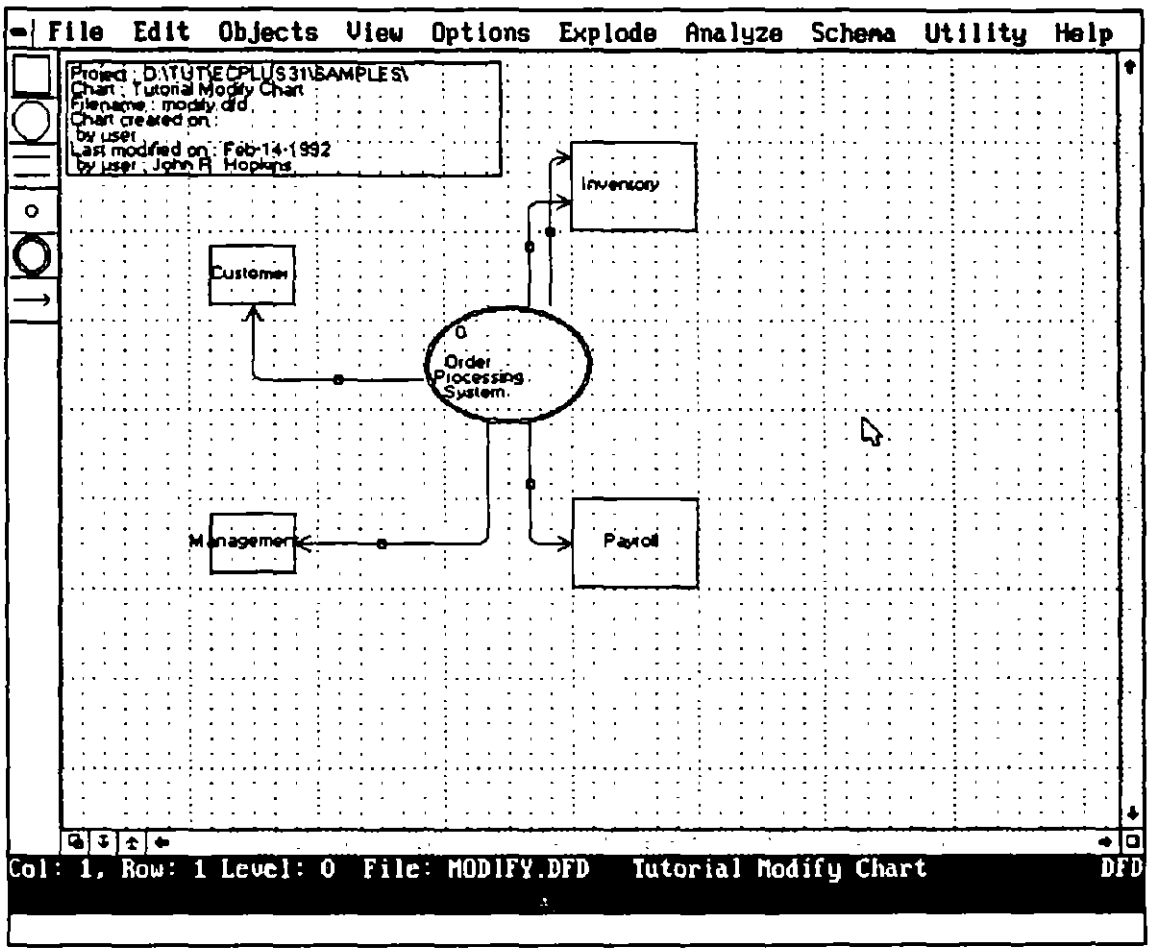


Figure 3.4 Chart with Moved Symbol

Copying a Symbol

Besides moving symbols, you can also duplicate them. This feature can be very useful when you are creating large and complex charts.

In this section, you will use the Copy command from the Edit menu to make a copy of the Management external entity symbol as follows:

1. Select the external entity symbol labeled "Management" by clicking on it. It will be highlighted.

2. Choose the Copy Symbol/Group command from the Edit menu.

3. Press and hold down the left mouse button. A small hand icon is displayed in the selected (Management) symbol.

4. While holding down the left mouse button, use the mouse to move the outline of the symbol to a position just below the original Management symbol as shown in Figure 3.5.

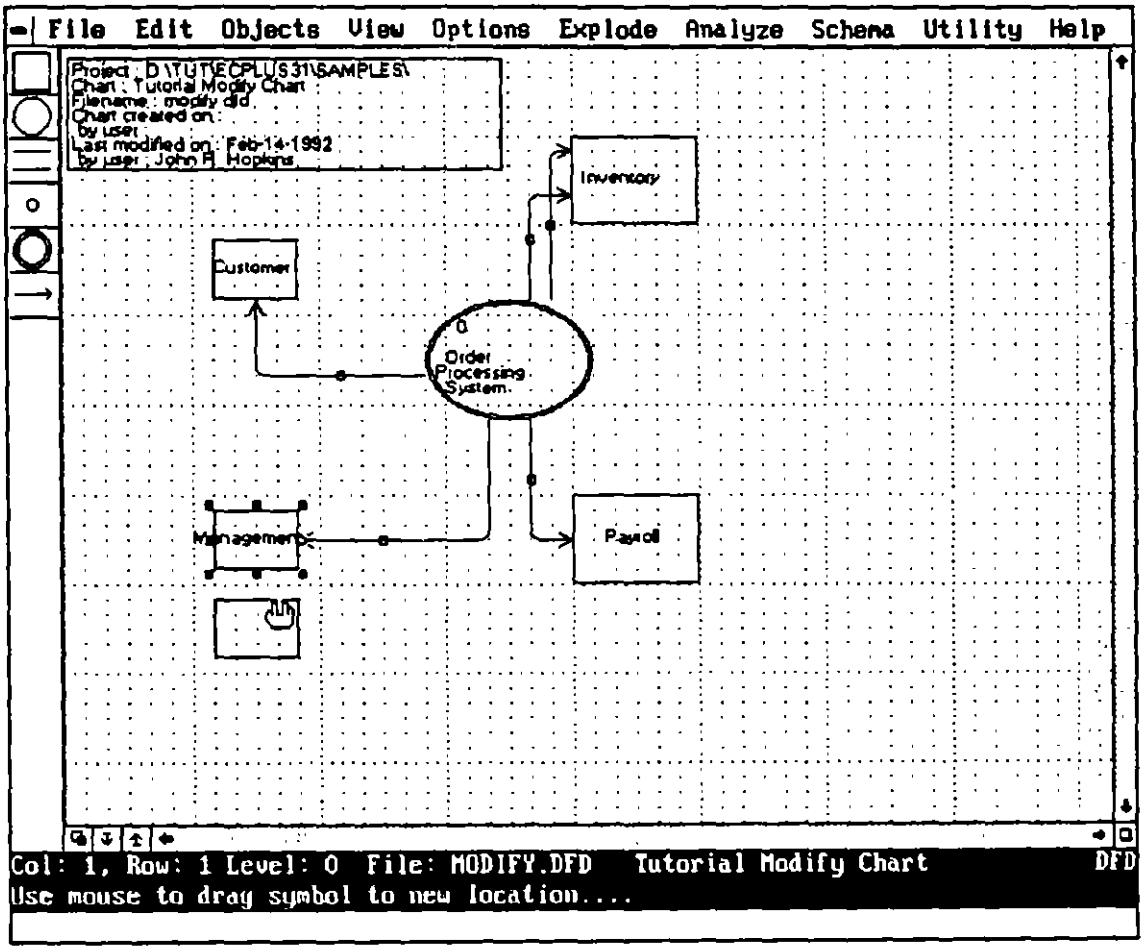


Figure 3.5 Copying the "Management" Symbol

5. Release the left mouse button.

   An exact copy of the Management symbol, and its label, is drawn at the new location.

Deleting a Symbol

It is a simple procedure to remove an object from the chart. Although it was a useful example, the new Management symbol that you just created is not necessary in this diagram.

In this section you will delete the symbol with its associated label and connection by using the Delete command from the Edit menu, as follows:

1. Select the new Management symbol by clicking on it. It will be highlighted.

2. Choose the 'Delete Object/Group' option from the Edit menu, or press the DEL key.

   A dialog box is displayed in the center of the screen, prompting you to confirm deletion of the selected symbol, as shown in Figure 3.6.

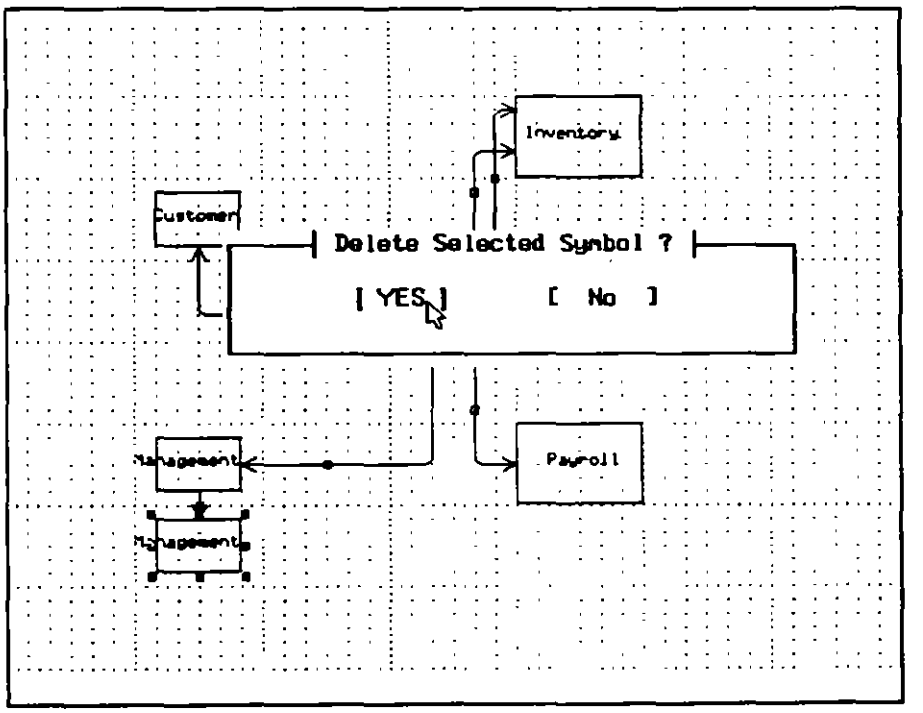


Figure 3.6 Prompt to Confirm Deletion of Selected Symbol

3. Click on 'Yes', or press the Y key.

4. The Management symbol along with its label, and the data flow connection to the original Management symbol, is deleted. If you get a dialog box requesting you to confirm deletion also of the Management entity data

dictionary entry, click on the No option button or press the ESC key to remove it.

5. Click on the Redraw icon to refresh the screen to remove any 'debris' resulting from the deletion.

Your chart should now look like figure 3.4 again.

Resizing a Symbol

You may decide that your chart can be clarified by changing the sizes of the symbols. The Customer and Management external entity symbols are smaller in size than they should be.

In this section you will resize these symbols by using the Change Symbol Size command from the Edit menu.

To resize the Customer and Management symbols:

1. Select the Customer symbol by clicking on it. It will be highlighted.

2. Select the 'Change Symbol Size' option from the Edit menu.

3. A pop-up list box containing the available symbol sizes is then displayed (see Figure 3.7). Available symbol sizes are: Small, Normal, Large, Extra Large, and Giant. The Small size entry is highlighted initially.

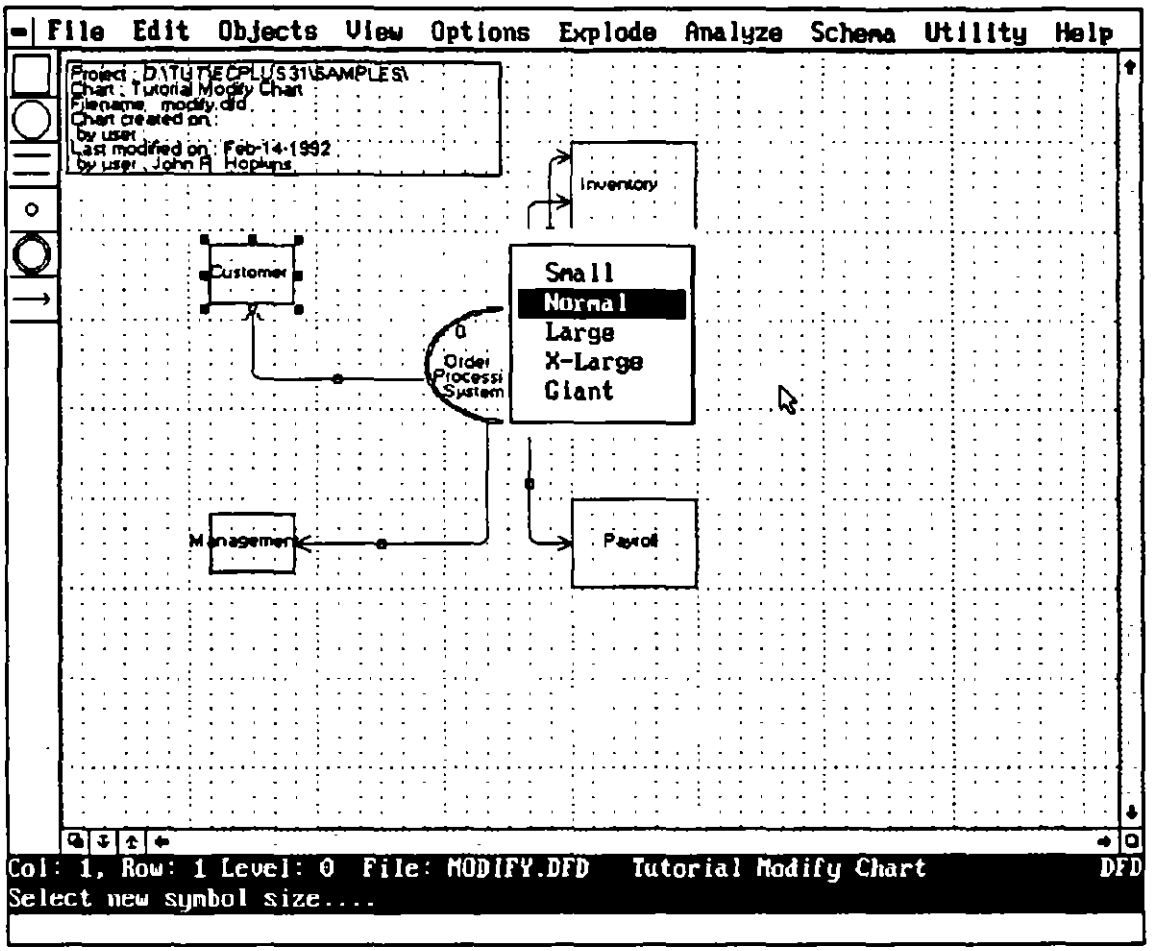


Figure 3.7 Select 'Normal' Entry Symbol Size from List

4. Select the Normal entry by clicking on it.

   The Customer symbol is then redrawn using this (larger) size.

5. Repeat Steps 1 to 4 for the Management symbol.

6. Click on the Redraw icon to refresh the screen.

Your diagram should look like that in Figure 3.8.

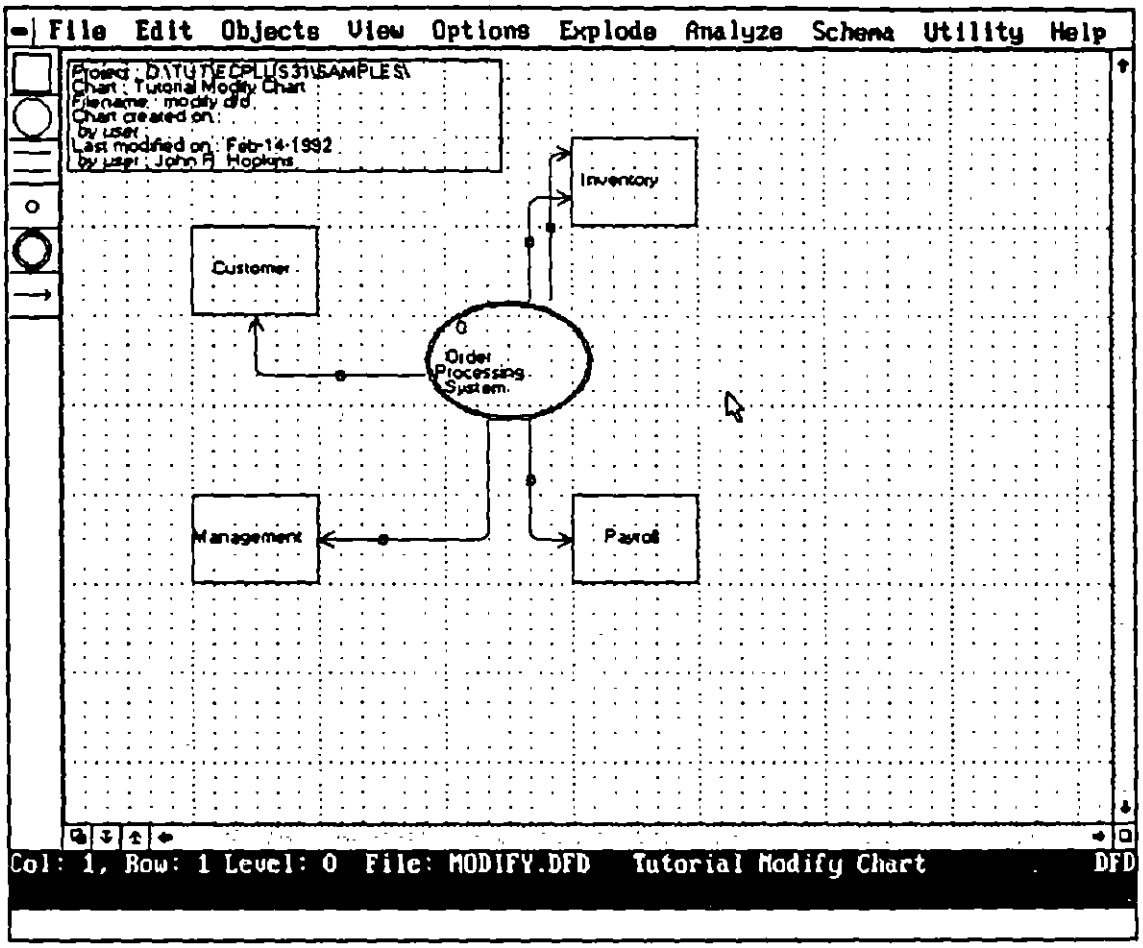


Figure 3.8 Resized Symbol

Changing a Symbol Label

You may need to change the label on a symbol. The procedure for doing so is similar to that used for creating a label, as you did in Lesson 2. Although the Inventory symbol is useful, it is wrong for this diagram.

In this section, you will use the Label Object command from the Edit menu to change the Inventory label as follows:

1. Click twice on the Inventory symbol to select its label, as shown in Figure 3.9.

2. A highlighted, dashed box is then displayed around the label text ("Inventory") showing that it is selected. If the symbol is selected, click on it again until only the label is selected.

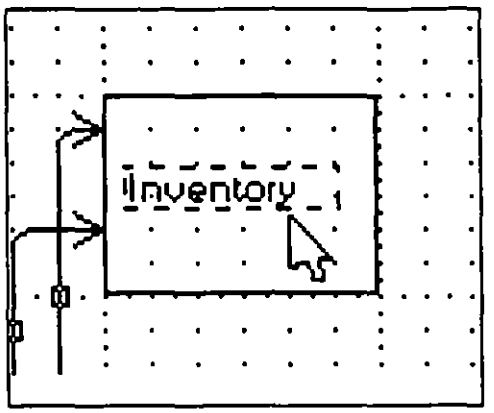


Figure 3.9 Selected Symbol Label

3. Select the 'Label Object' option from the Edit menu, or press ALT + L.

4. A text entry box is displayed in the center of the screen containing the current label for the symbol.

5. Press CTRL + Y to delete the current line of text.

6. Type in the new label "Supplier" (see Figure 3.10).

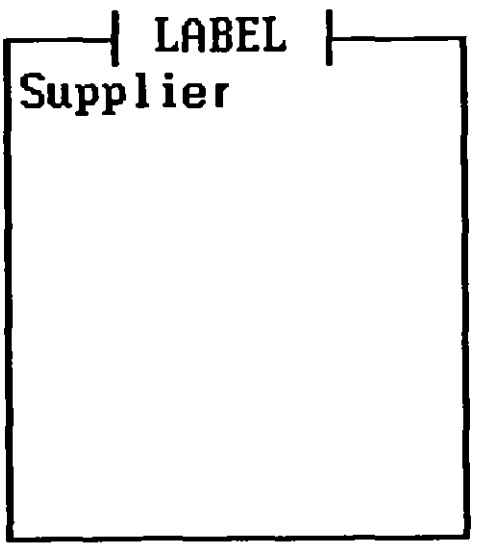


Figure 3.10 Label Text Entry Box

7. Press the F3 key.

8. The new text for the label will appear at the current label position inside the symbol, as shown in Figure 3.11. You do not need to reposition the label. Press the ENTER key, or click the right mouse button, to accept the default label position.

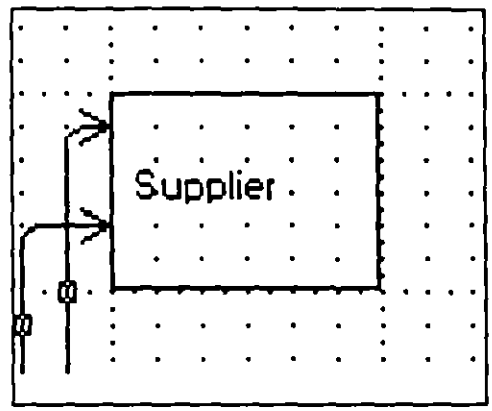


Figure 3.11 Label Positioned Inside Symbol

Changing a Symbol Type

You can easily change the type of a symbol as you develop your chart. You may have noticed that a Multi-Data Process symbol type has been (wrongly) used for the Order Processing System. This should be a regular Data Process symbol.

In this section, you will use the Change Object Type command from the Edit menu to correct it.

To change the symbol type for the Order Processing System:

1. Select the Order Processing System symbol by clicking on it (click near the center of the symbol). It will be highlighted.

2. Select the 'Change Object Type' option from the Edit menu.

3. A list box will pop up in the center of the screen containing the available symbol types for the current (DFD) chart type, as shown in Figure 3.12. The 'Multi-Data Process' entry is highlighted initially.

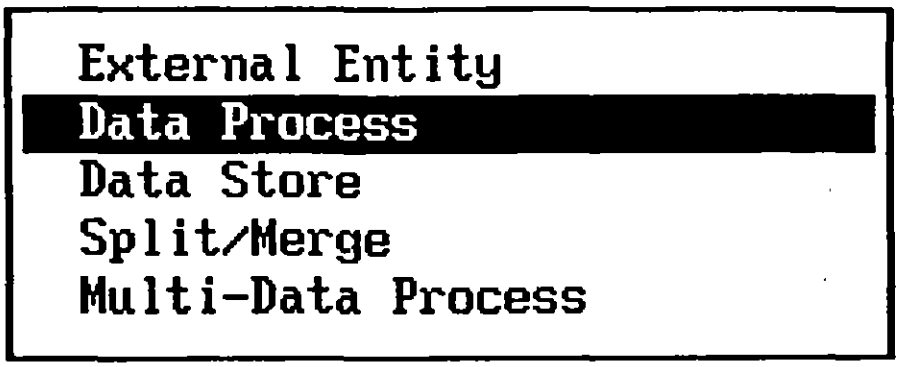


Figure 3.12 List of Available Symbol Types for a DFD

4. Select the 'Data Process' option by clicking on it.

   The symbol is redrawn using the new symbol type (without affecting the current label).

Moving a Group of Objects

The Supplier and Payroll external entity symbols are both too close to the Order Processing System symbol. Rather than moving these independently, you can move them both together as a single unit (group).

In this section, you will use the Move command to move these symbols together as a group.

To move the Supplier and Payroll external entity symbols together as a group:

1. Select the Supplier and Payroll symbols together as a group by drawing a selection box around them, as described in Lesson 1. A dashed selection box appears around the Supplier and Payroll symbols, both of which will be highlighted.

2. Position the mouse pointer over the selected group of symbols, inside the selection box.

3. Press and hold down the left mouse button. A small hand icon is displayed (see Figure 3.13).

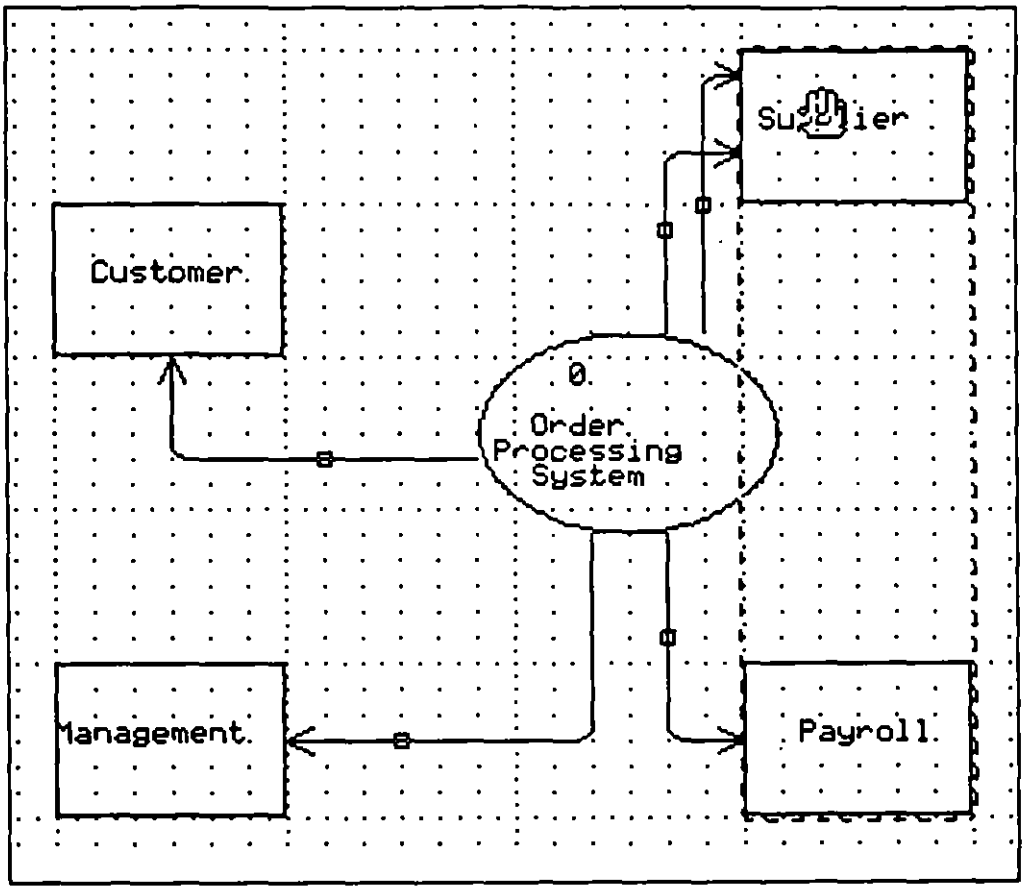


Figure 3.13 Hand Icon Positioned to Move the Selected Group of Symbols

4. While holding down the left mouse button, use the mouse to drag the dashed box one column (grid square) to the right.

5. Release the left mouse button.

6. The group of selected symbols is placed at the new location. Notice that the associated connections and labels are also relocated.

7. Click the right mouse button to de-select the Supplier and Payroll symbols and erase the selection box.

8. If necessary, click on the Redraw icon to remove any 'debris' resulting from movement of the selected symbols.

Your chart should now look like this:

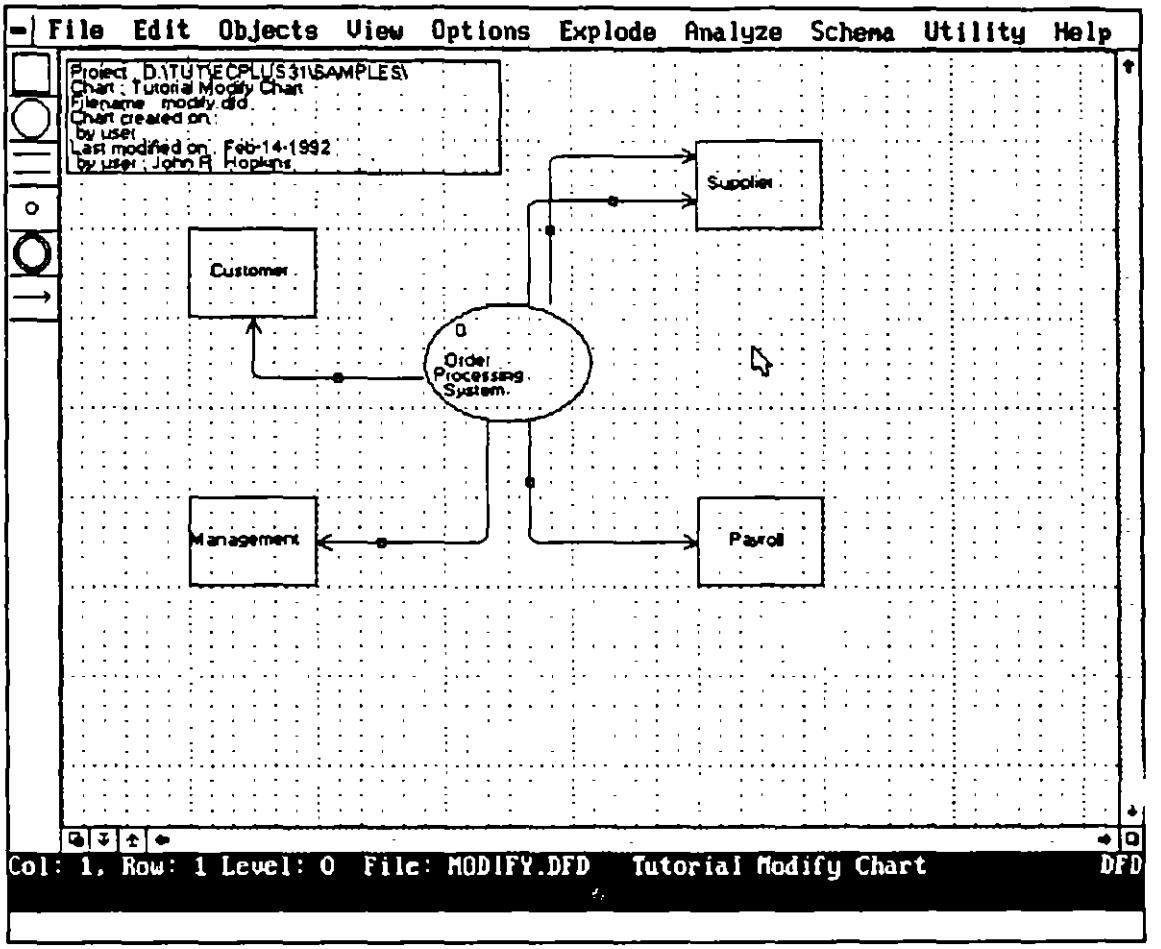


Figure 3.14 Chart with Moved Group of Symbols

Copying a Group of Objects

Earlier in this lesson, you duplicated a single symbol (using the Copy Symbol command). You can also duplicate a group of symbols, rather than creating or copying the symbols independently. This process is similar to the Move Object/Group procedure just described.

In this section, you will use the Copy command.

To copy the Supplier and Payroll symbols together as a group:

1. Select the Supplier and Payroll symbols together as a group, just as you did for the Move Block procedure (see steps 1 to 3 of "Moving a Group of Symbols"). A dashed selection box appears around the highlighted symbols.

2. Choose the 'Copy Symbol/Group' option from the Edit menu.

3. Press and hold down the left mouse button. A hand icon appears (see Figure 3.15).

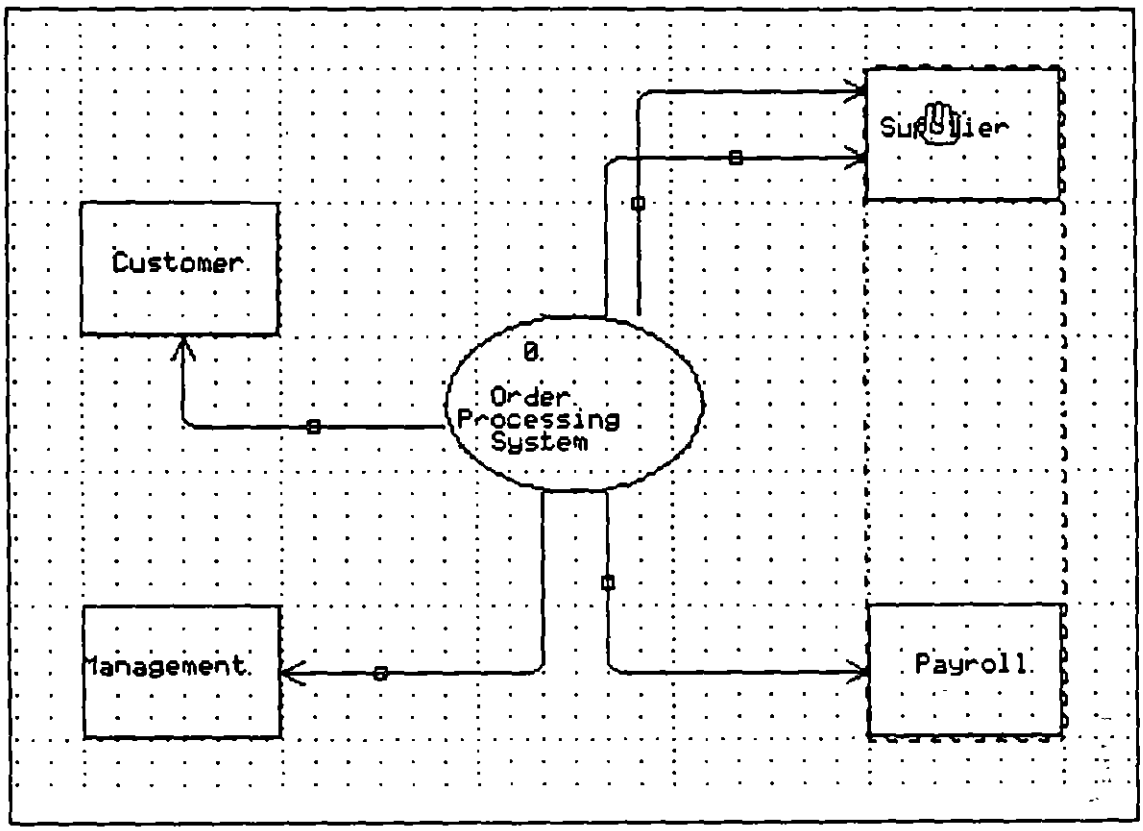


Figure 3.15 Hand Icon Positioned to Copy Symbol Group

4. Use the mouse to drag the dashed selection box two columns (two grid squares) to the right of the symbols' current position.

5. Release the left mouse button.

6. A copy of the two selected symbols is placed at the new location.

7. Click the right mouse button to de-select the Supplier and Payroll symbols.

When you have finished, your chart should now look like this:

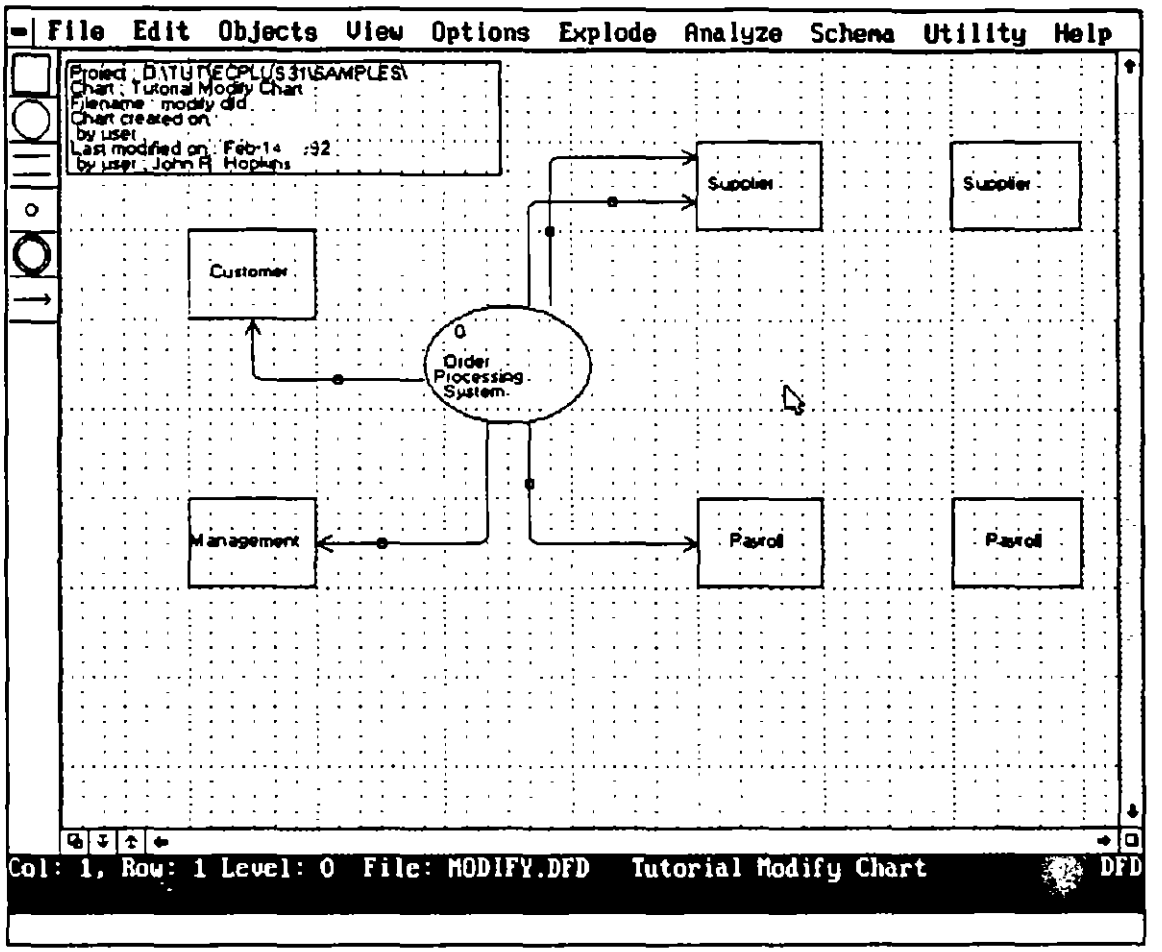


Figure 3.16 Chart with Copied Group of Symbols

Deleting a Group of Objects

Any modifications you can make to a single object, you can also make to a group of objects. This is also the case for delete operations. Rather than incorporating the extraneous, duplicate Inventory and Payroll symbols into the chart, or deleting them individually, delete both of the new symbols in a single operation.

In this section, you will use the Delete Group command.

To delete the new (right-most) Supplier and Payroll symbols together as a group:

1. Select the Supplier and Payroll symbols together as a group (just as you did for the Move Group and Copy Group procedures).

2. Select the 'Delete Object/Group' option from the Edit menu, or press the DEL key.

3. You are prompted to verify that you want to delete any Data Dictionary Entries (DDE) associated with the symbols as they are deleted. EasyCASE Plus automatically displays this prompt whenever you delete a symbol or group of symbols, whether or not there are existing Data Dictionary Entries for the symbols. At this point in the tutorial, you have not yet linked chart objects to entries in the data dictionary. Hence, this dialog box is irrelevant at this point.

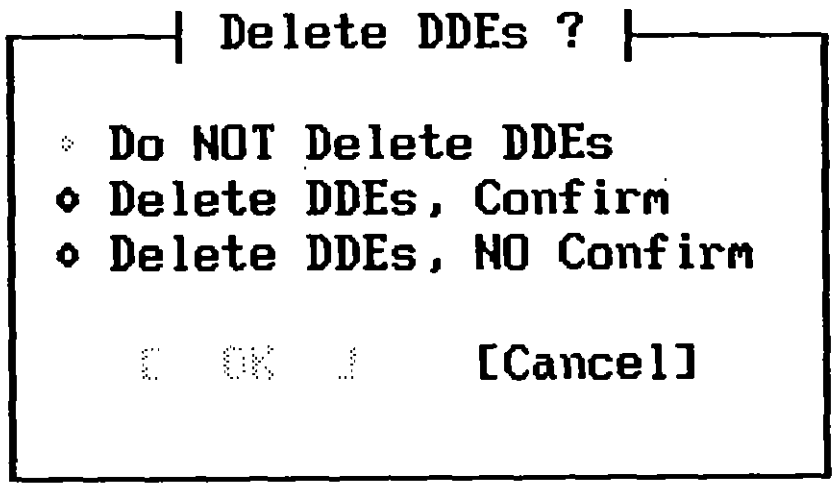


Figure 3.17 Prompt for Deletion of Associated Data Dictionary Entries

4. Click on the 'Do NOT Delete DDEs' option.

5. Click on 'OK.'

6. The selected symbols, and their labels, are erased from the chart.

Your chart should now look like figure 3.14 again.

Modifying Connections

Whenever the logic for the diagram changes, you can change the data flow to accommodate it. EasyCASE Plus gives you a number of ways to modify connections:

- Change the direction of a connection

- Change the end point positions of a connection (i.e. where it connects with a symbol)
- Reroute the connection (i.e. changing the path of the connection)
- Change the destination and/or source symbols for a connection
- Delete a connection
- Change the number of arrowheads or symbols on the end(s) of a connection

Every connection has a small box at the mid-point of a line segment. This is the "handle" you will use to select the connection. When a connection is selected, a marker is displayed at each corner and mid-point of the line segments that constitute the connection, as well as at each end. These markers function in two ways:

- As a select point to grab and move the connection line segments
- As a select point for removing and possibly rerouting sections of the connection

The following exercises will illustrate how to modify connections.

Changing the Direction of a Connection

Changing the direction of a data flow is quick and easy. For example, the direction of the connection between the Order Processing System symbol and the Customer symbol is incorrect and needs to be reversed.

In this section, you will use the Change Direction command from the Edit menu to reverse the direction of the connection, as follows:

1. Select the data flow connection that originates from the Order Processing System symbol and enters the Customer symbol (click on the connection's handle). The connection is highlighted and markers are visible at various positions on it.

2. Select the 'Change Direction' option from the Edit menu.

3. The data flow is reversed so that it represents a flow of data from the Customer external entity symbol into the Order Processing System data process symbol.

When you are finished, the Customer and Order Processing connections should look like this:

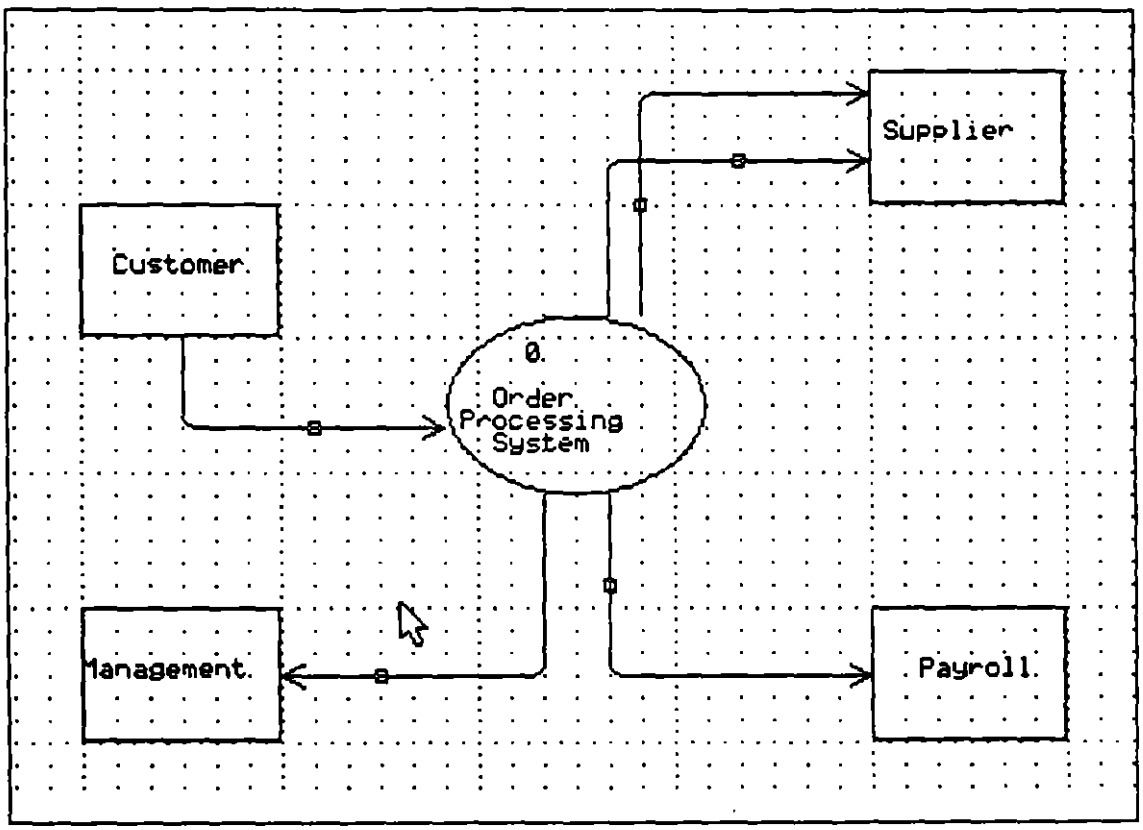


Figure 3.18 Chart with Flow Direction Changed

Deleting a Connection

You can delete a connection just as you would a symbol. As an illustration, delete one of the connections to the Supplier symbol.

You will use the Delete command from the Edit menu to accomplish this.

To delete the connection:

1. Select the lower of the two data flow connections terminating in the Supplier symbol. The connection is highlighted and markers appear along it.

2. Select the 'Delete Object/Group' option from the Edit menu, or press the DEL key.

3. A dialog box pops up in the center of the screen requesting you to confirm deletion of the selected symbol (see Figure 3.19).

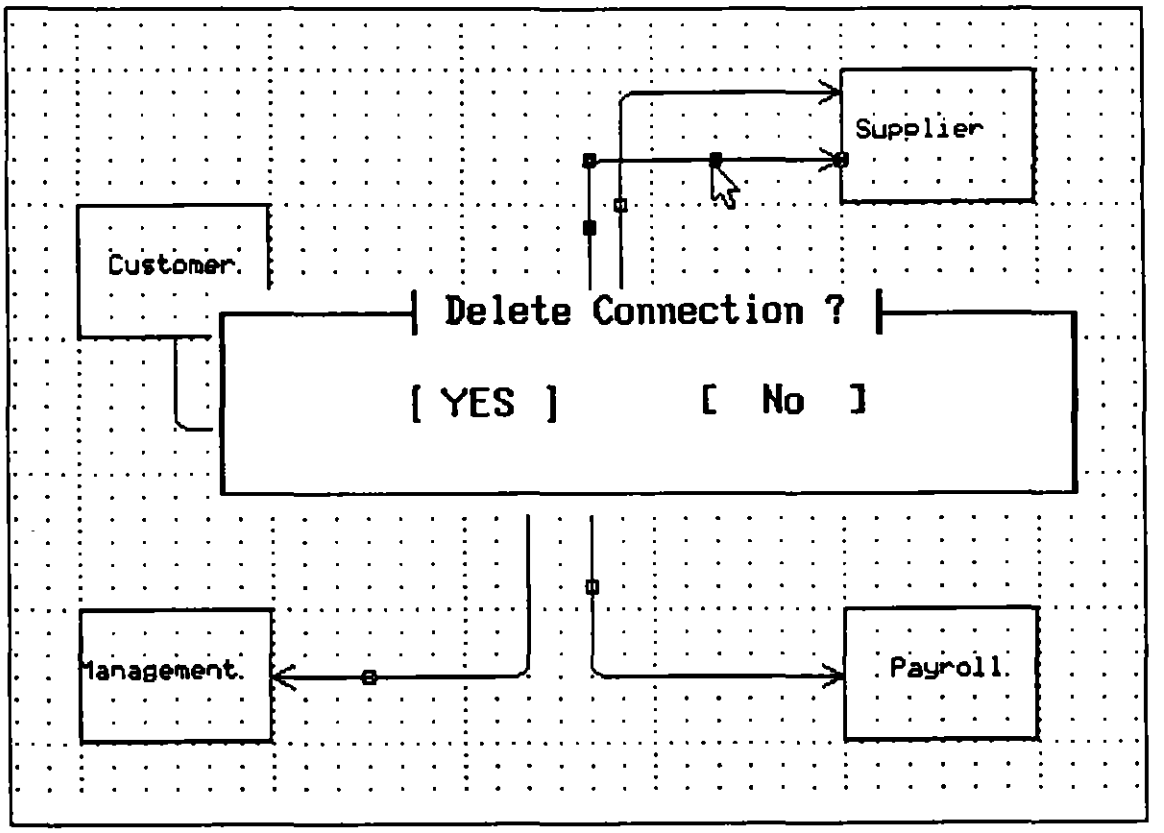


Figure 3.19 Prompt to Confirm Deletion of Selected Connection

4. Click on the 'Yes' button, or press the Y key.

5. The connection is erased from the chart (see Figure 3.20).

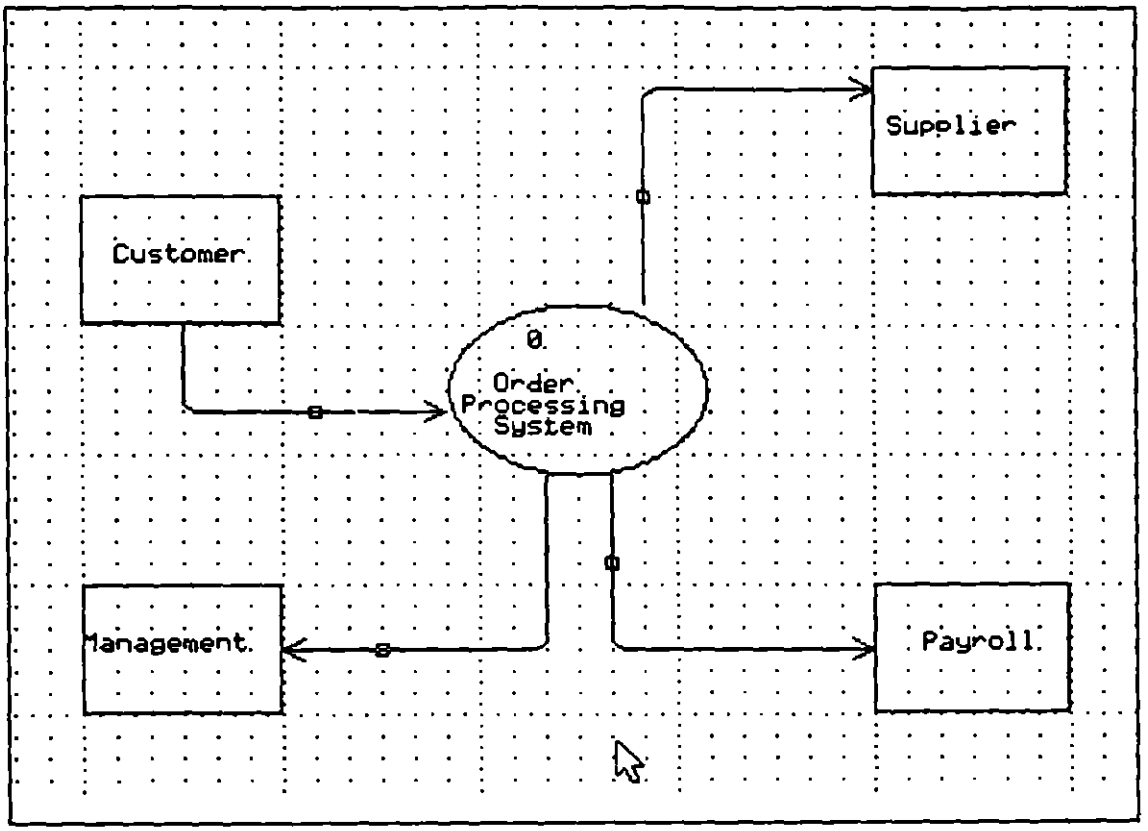


Figure 3.20 Chart with Connection Deleted

Changing the End Point for a Connection

You can relocate the end of a connection, or even both ends, to a different point on the symbol, or even to another symbol. In the example diagram, the end points for the connections between the Order Processing System symbol and the Supplier symbol are wrong at the Supplier end. In this section, you will change the connection's end points on the Supplier symbol to a slightly different position on the perimeter of that symbol to clarify the data flow.

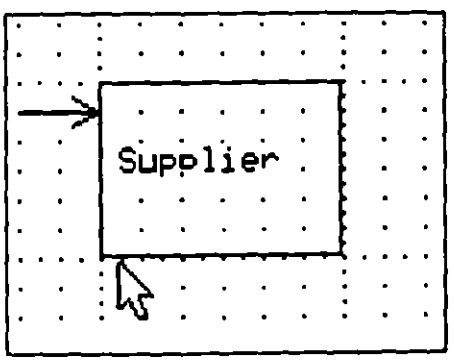


Figure 3.21 Poor Connection End Point at Symbol

To change the connection end point at the Supplier external entity symbol:

1. Select the data flow connection that is an input to the Supplier symbol. The connection is highlighted with all of its markers displayed. You will be accessing one of the markers at the end of the connection.

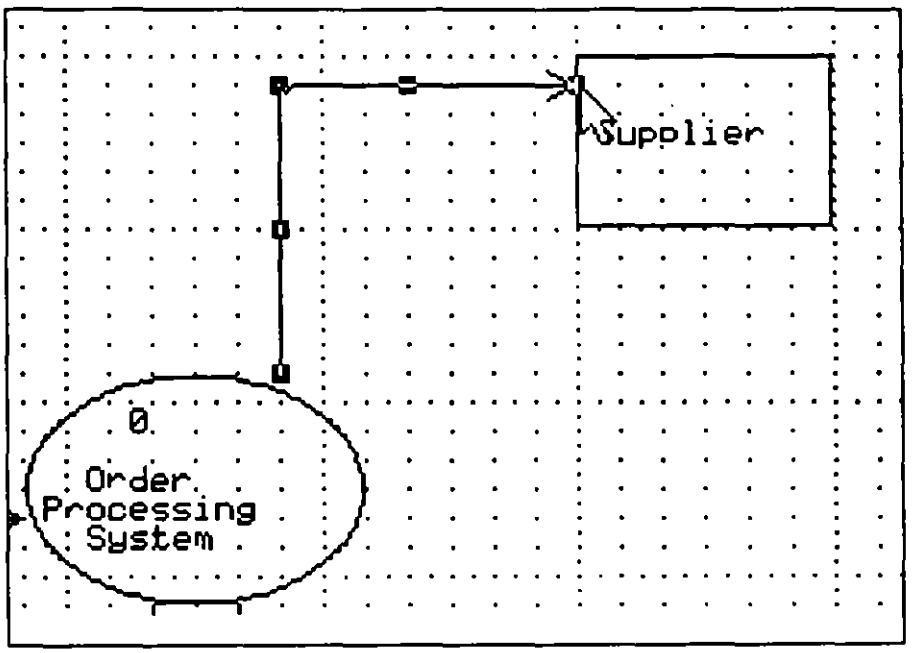


Figure 3.22 Selected Connection

2. Click on the marker box located at the connection's end point, coincident with the Supplier symbol as shown in Figure 3.22.

3. The preceding section of the connection is removed (from the selected end point back to the last change in direction). The marker moves to the new end point, as shown in Figure 3.23.

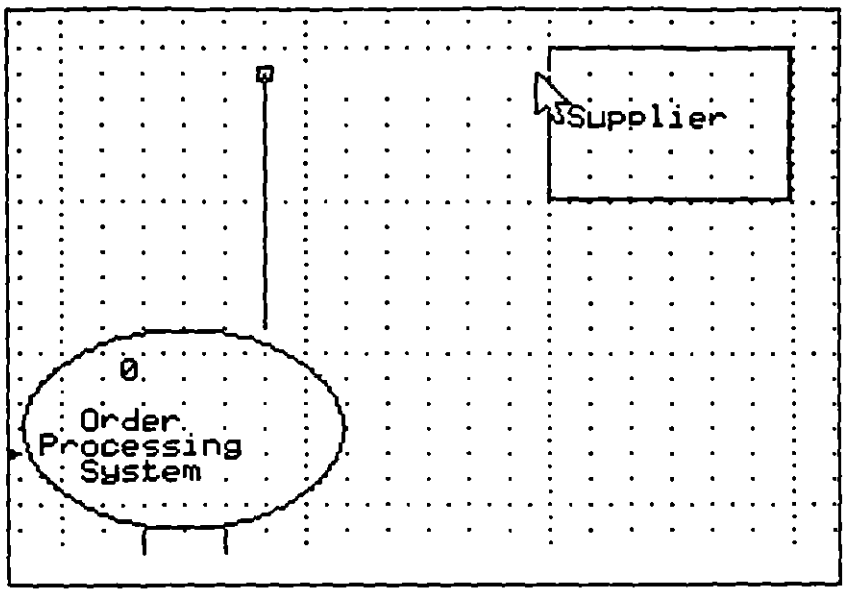


Figure 3.23 Connection with Last Segment Removed

4. Click the right mouse button once to remove the other line segment.

5. Select the Supplier symbol by clicking on it.

6. The allowed end points for connection on the perimeter of the symbol are indicated by markers, as shown in Figure 3.24.

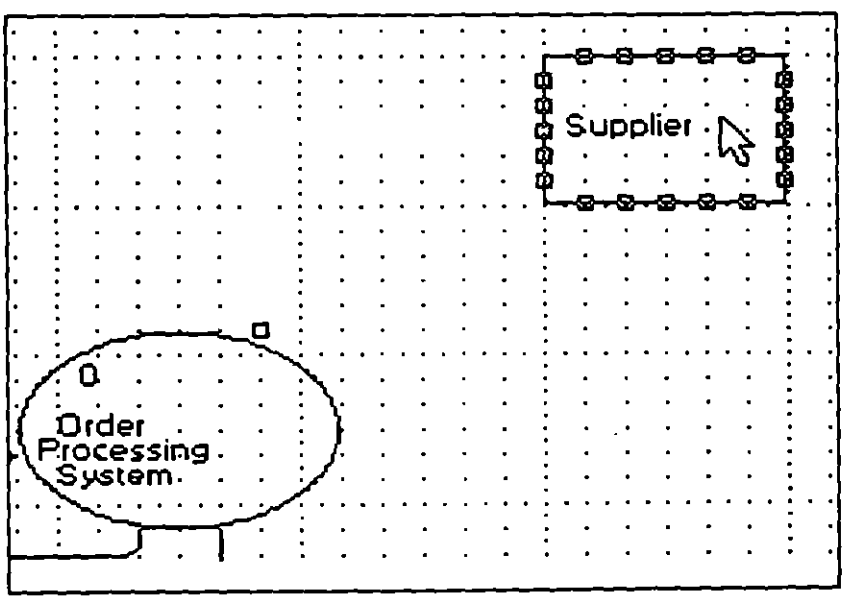


Figure 3.24 Available Entry Ports at Destination Symbol

7. Click on the marker located half way down the left side of this symbol.

8. The connection is then rerouted to this new end point.

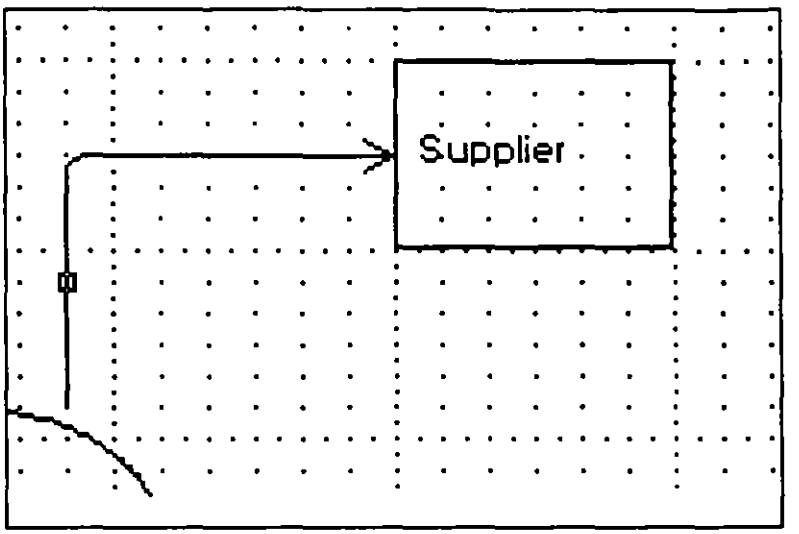


Figure 3.25 New Placement of Flow

Rerouting a Connection

Although connections between symbols are automatically routed, sometimes the routing used for a connection may conflict with other chart objects. To illustrate this, you will add a new External Entity symbol just below the Order Processing System symbol. You will then reroute the affected data flow inputs for the Management and Payroll symbols so that they no longer pass through the new symbol.

First, add a new external entity symbol that conflicts with the Management and Payroll data flows as follows:

1. Choose the 'External Entity' option from the Objects menu.

2. Click on the grid square midway between the Management and Payroll external entity symbols, as shown in Figure 3.26.

3. An external entity symbol is placed at that position on the chart, on top of the connections between the Order Processing System symbol and the Management and Payroll symbols, as shown in Figure 3.26.

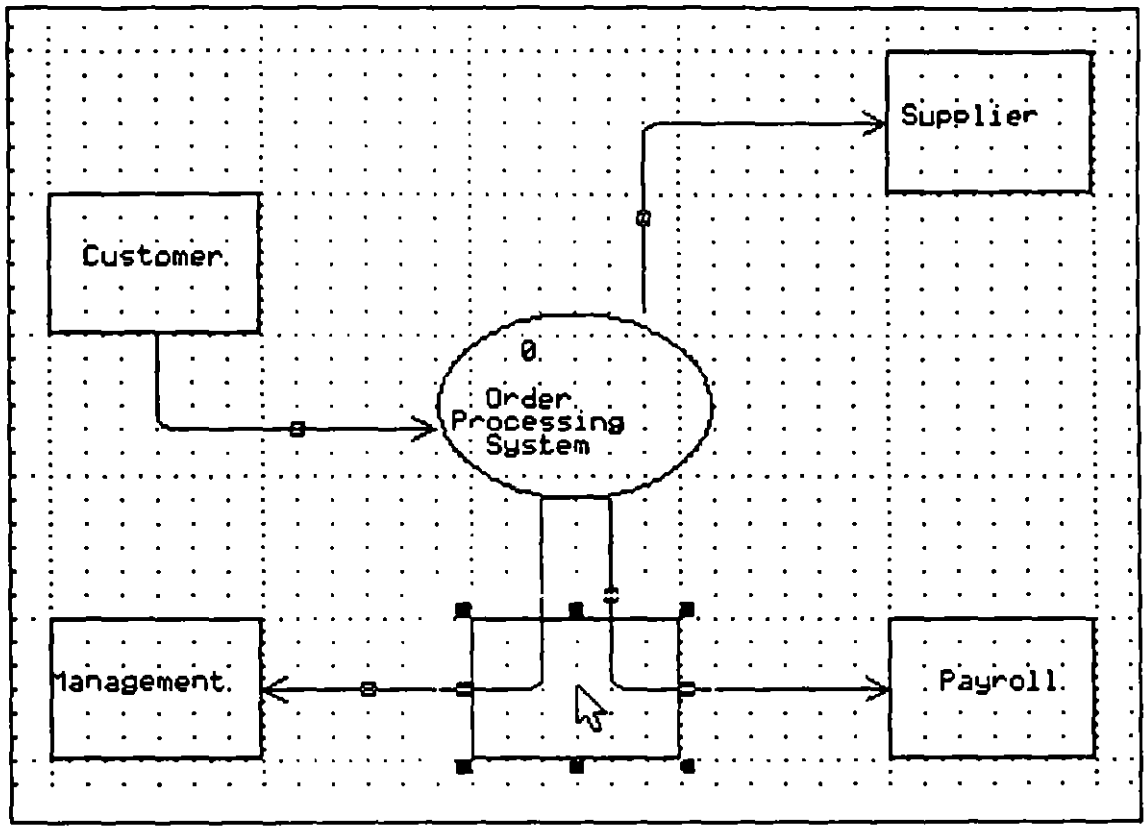


Figure 3.26 Placement of Extra External Entity

Next, reroute the affected Management and Payroll connections:

1. Select the Payroll data flow (click on the small box or "handle" located on the connection). The connection is highlighted and its markers are displayed.

2. Click the left mouse button on the connection end point marker at the Payroll symbol end.

   The last line segment is removed from the connection at the selected end and the end point marker moves to the new end point position, as shown in Figure 3.27.

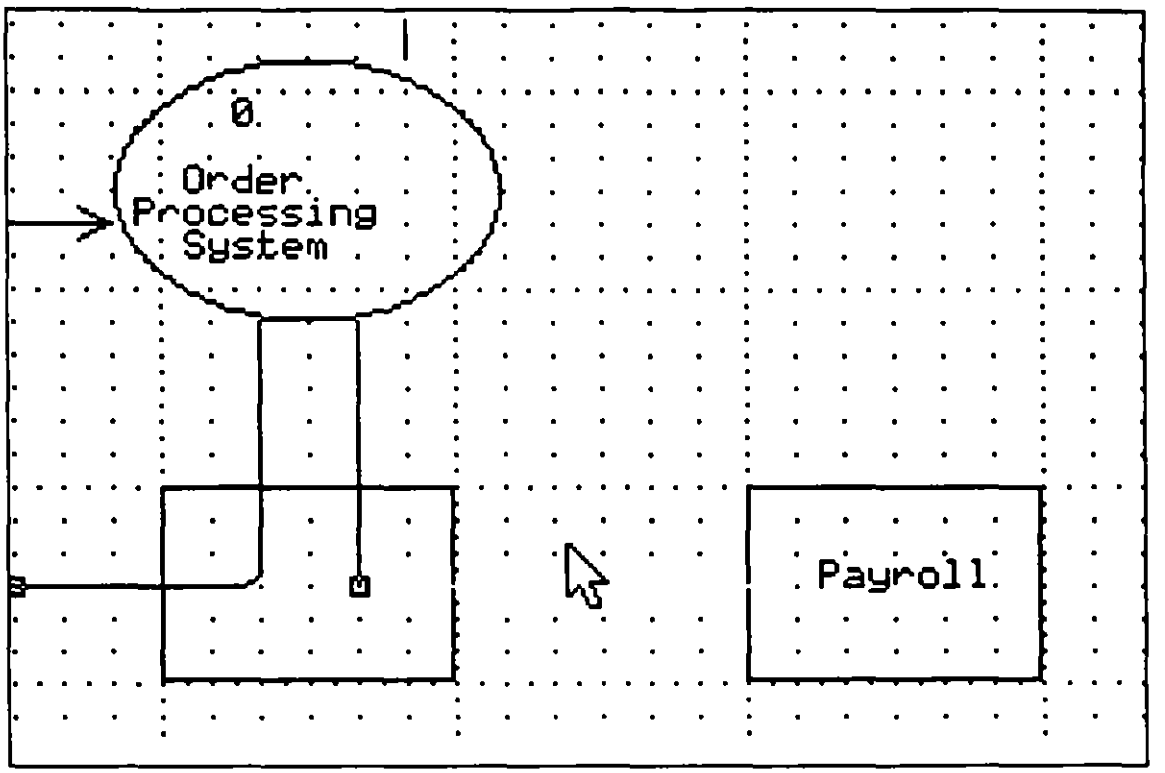


Figure 3.27 Connection with Line Segment Removed

3. Click the right mouse button, or press the BACKSPACE key, once only.

   The remaining line segment is removed from the connection. There are no line segments left on the connection.

4. Click the left mouse button inside the Payroll symbol.

   Markers are displayed around the Payroll symbol to indicate allowable entry positions for the connection.

5. Click on the marker located midway along the top of the Payroll symbol.

   The connection is rerouted to the new end point, avoiding the external entity symbol, as shown in Figure 3.28.

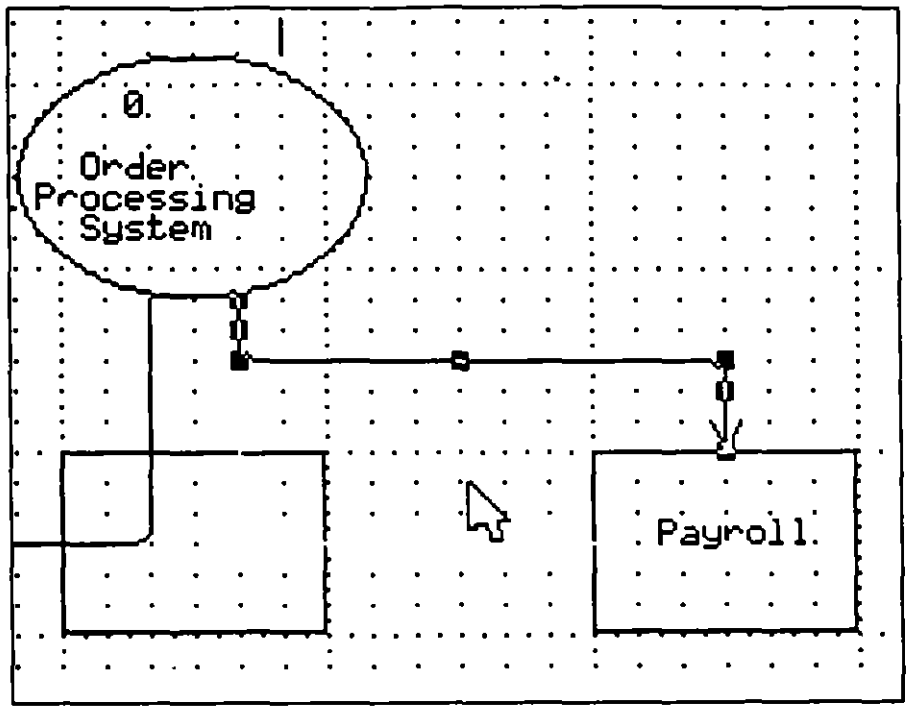


Figure 3.28 Payroll Symbol with Connection Moved

6. Repeat Steps 1 through 5 for the Management data flow.

The resulting connections should look like this:

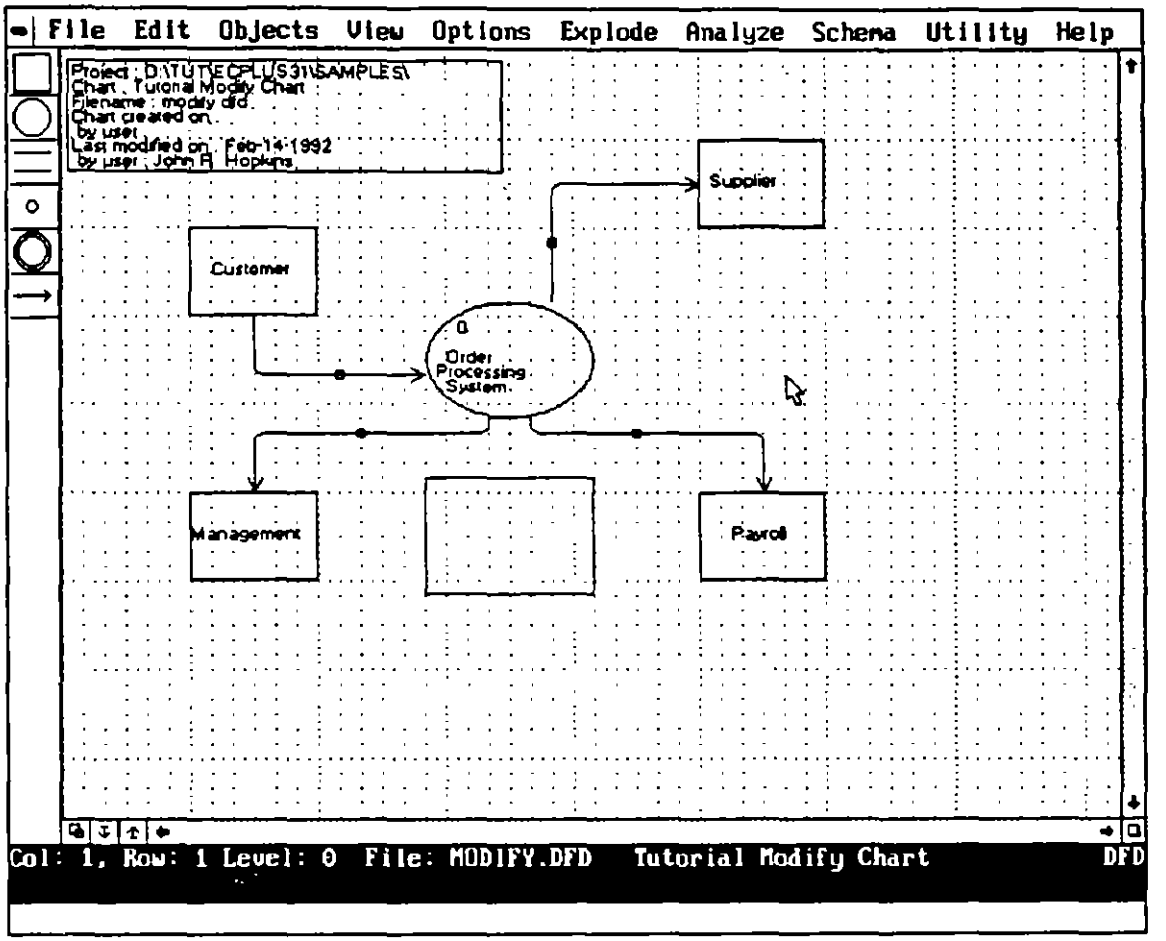


Figure 3.29 Manually Rerouted Connections

Changing the Destination Symbol for a Connection

Besides changing a connection's end points on a particular symbol, as you did for the Supplier symbol, you can change the destination symbol for a connection. In this section, you will try this by moving the destination for the data flow input connection from the Payroll symbol to the new symbol.

To change the destination of the data flow from the Payroll symbol to the new symbol:

1. Select the Payroll data flow (click on the small box or "handle" located on the connection). The connection is highlighted and all of the markers are displayed.

2. Click on the connection end point at the Payroll symbol end.

The last line segment is removed from the connection.

3. Click the right mouse button, or press the BACKSPACE key, once only.

   Another line segment is removed from the connection, as shown in Figure 3.30.

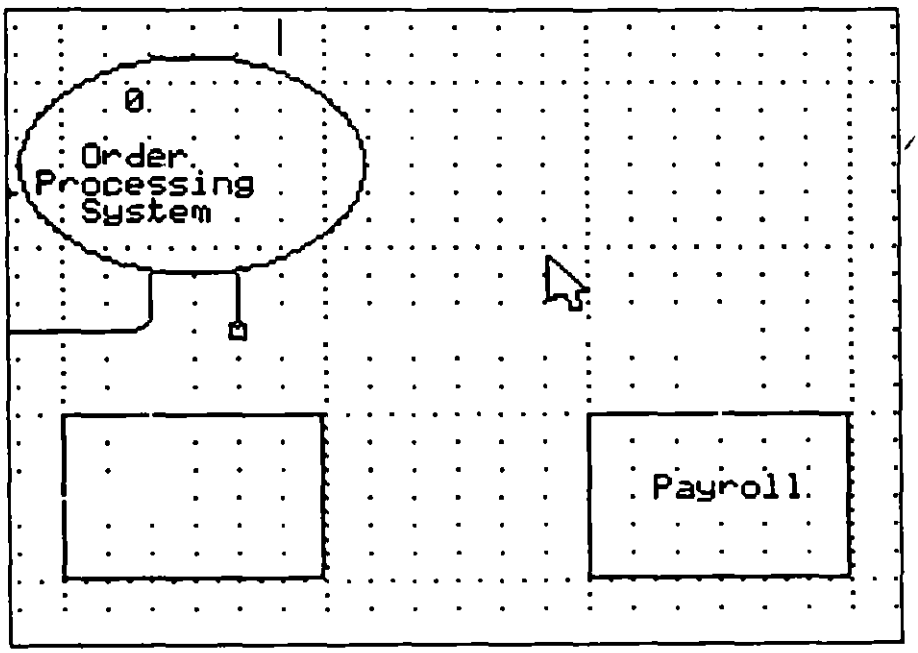


Figure 3.30 Connection with Line Segment Removed

4. Click the left mouse button inside the new symbol.

   The end points are displayed by marker boxes.

5. Click on the end point marker located on the top of the new symbol that is directly below the source end point on the Order Processing System symbol.

   The connection is rerouted to the new symbol destination, as shown in Figure 3.31.

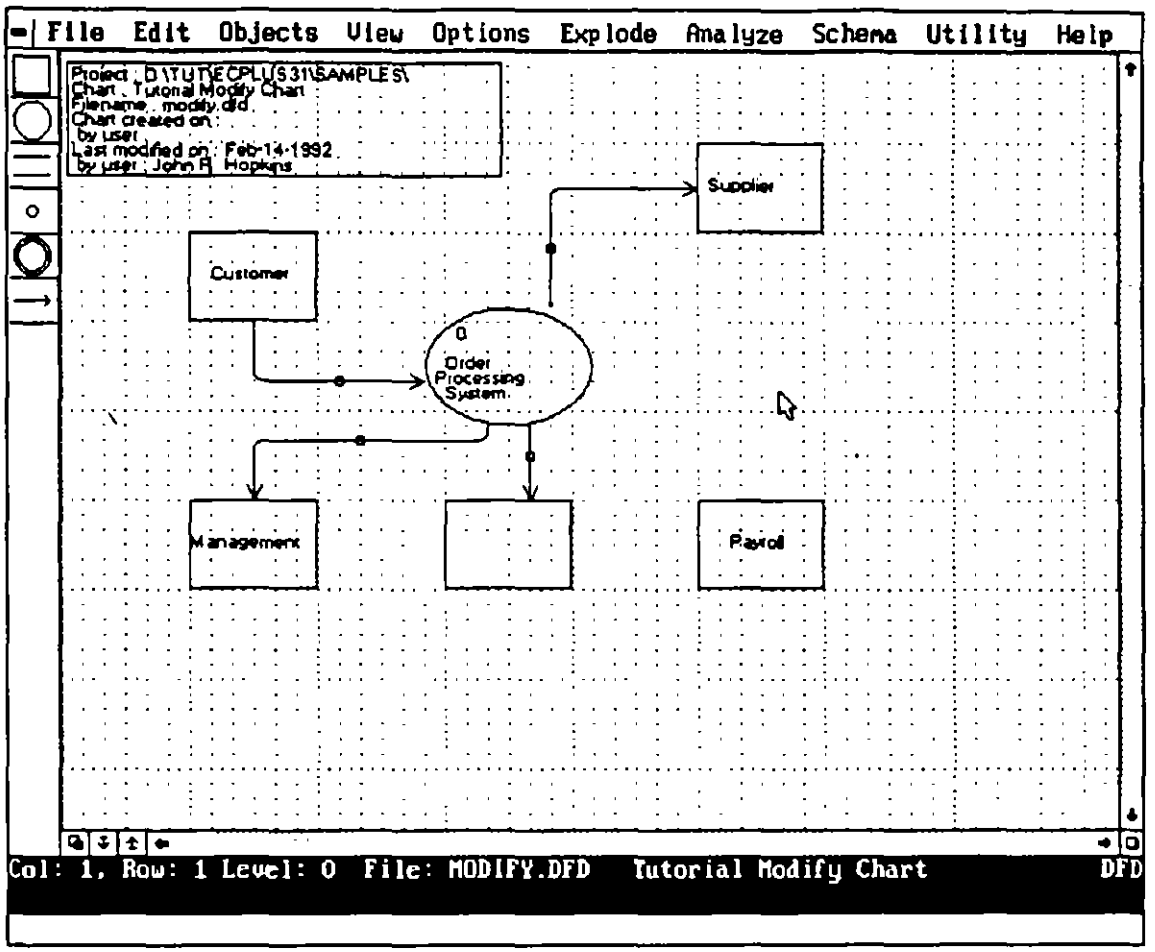


Figure 3.31 Rerouted Connection Destination

Changing the Number of Arrowheads on a Connection

You can change the number of arrowheads used for a specific connection, or for all connections. The Default # Arrowheads setting in the Options menu affects all connections. You use the Change # Arrowheads command in the Edit menu to change the number of arrowheads for a particular connection.

To change the number of arrowheads for the connection between the Supplier external entity and Order Processing symbols:

1. Select the connection between the Supplier and the Order Processing System symbols by clicking on its "handle."

   The connection is highlighted, and its markers are displayed. You will observe that this connection currently has a single arrowhead on it at the Supplier end.

2. Select the 'Change # Arrowheads' option from the Edit menu.

   A dialog box is displayed listing the available arrowhead characteristics, as shown in Figure 3.32.

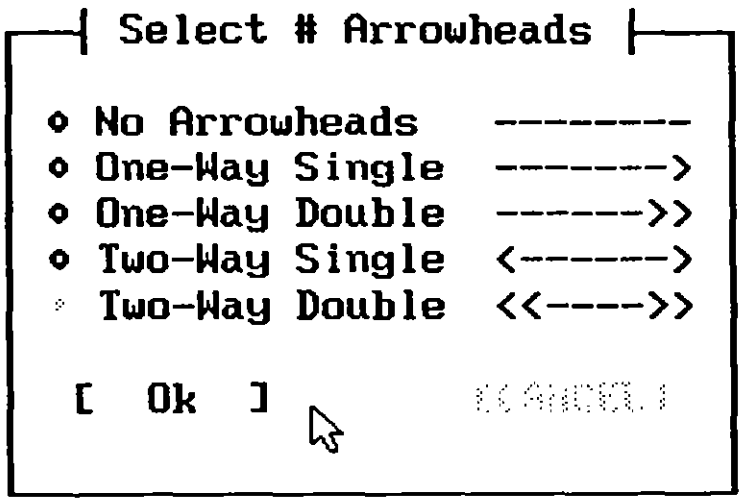


Figure 3.32 Number of Arrowheads Options Dialog Box

3. Click on the 'Two-Way Double' setting.

4. Click on the 'OK' button.

5. The Supplier connection is drawn with double arrowheads at each end of it.

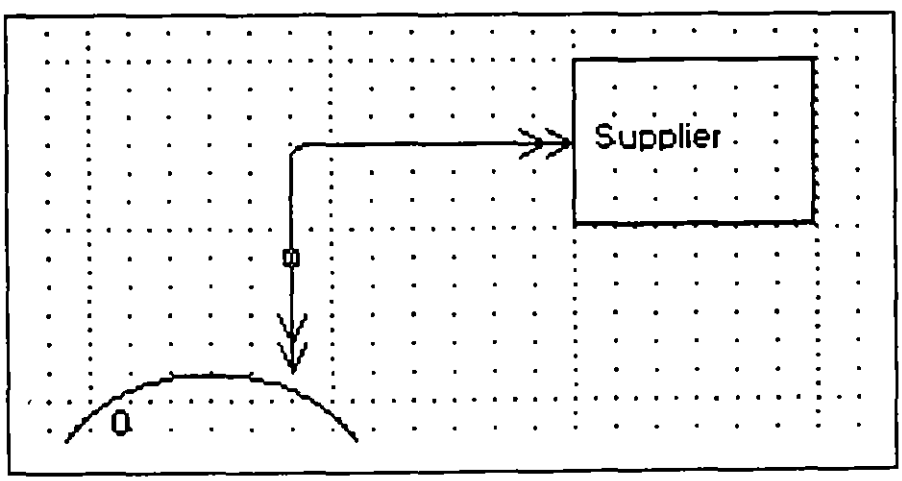


Figure 3.33 Connection Drawn with Double Arrowheads at Each End

Moving the Entire Chart

While you are developing a chart, you need not be overly concerned about its position in the chart window. You can easily change this by moving all of the chart objects together as a single unit. For example, the only remaining problem with the MODIFY chart is its position in relation to the Title Block.

In this section, you will use the Move Chart command to reposition the chart.

To move the entire chart as a unit:

1. Select the Move Whole Chart command from the Edit menu.

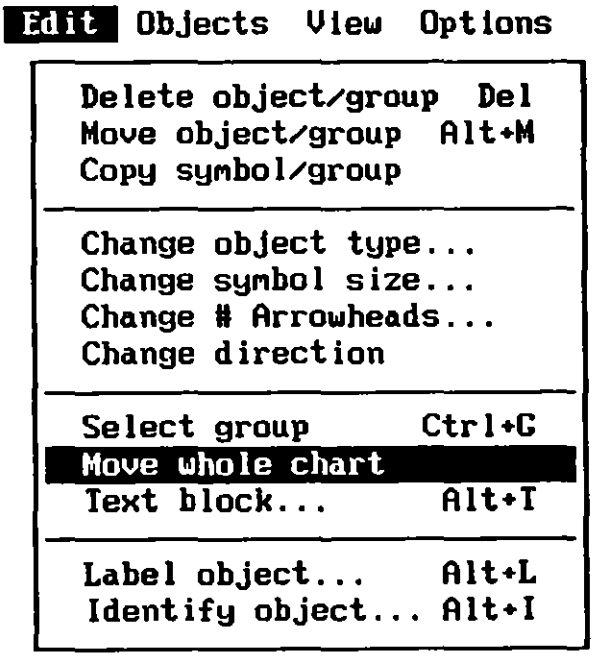


Figure 3.34 Edit Menu containing the Move Whole Chart Command

This automatically zooms the chart window to full chart mode. The entire chart is visible and selected, shown by a dashed box outlining all of the objects (except the title block) on the chart.

2. Press and hold down the left mouse button.

3. Still holding down the left mouse button, drag the selection box one grid square down and to the right of the chart's present position (as shown in Figure 3.35).

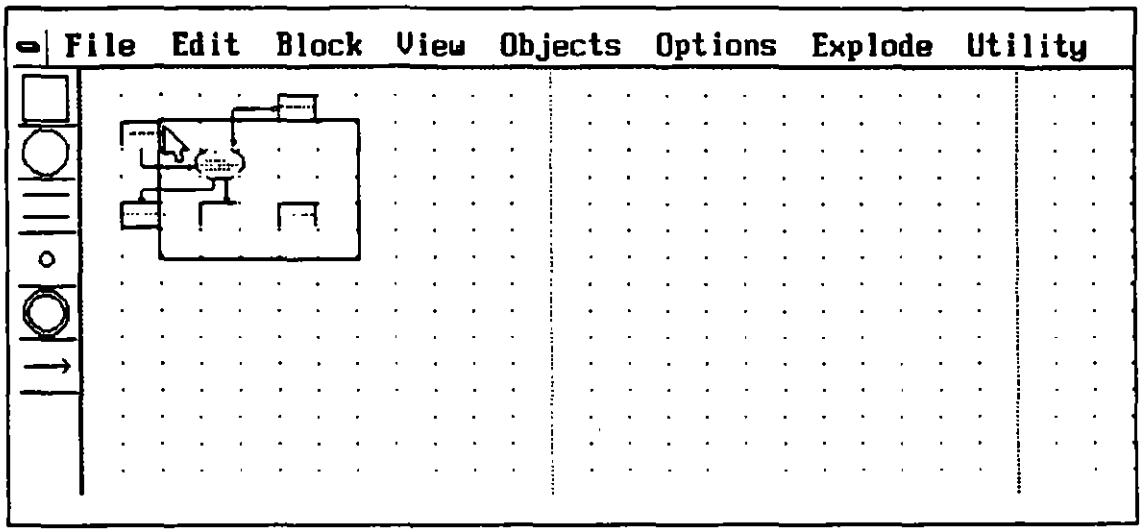


Figure 3.35 Selection Box 'Dragging' Chart Outline to New Location

4. Release the left mouse button.

5. A dialog box pops up in the center of the screen, prompting you to confirm the placement of the chart.

6. Click on the 'OK' option button to confirm the new position and re-display the chart at Actual size.

Your chart is now complete and, aside from possible spacing differences, should look like this:

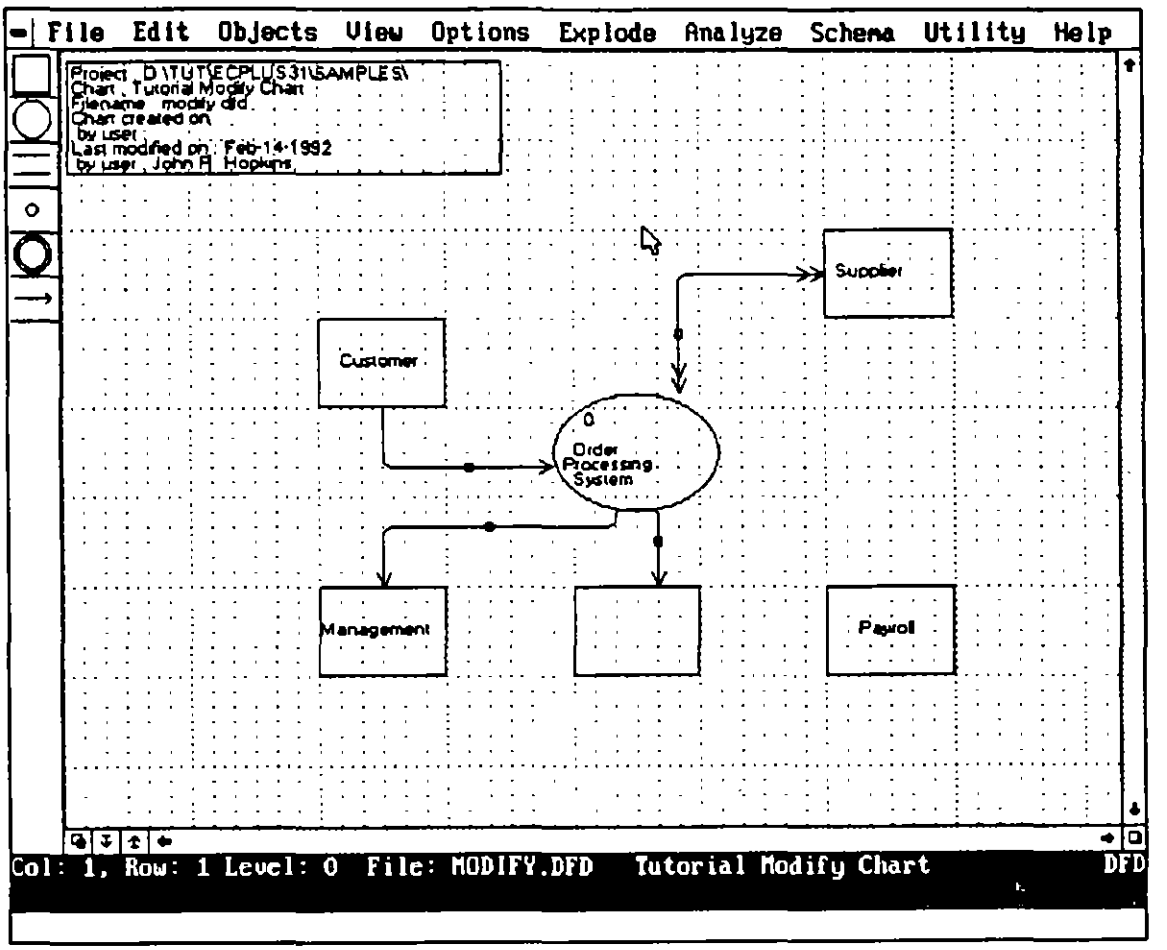


Figure 3.36 Chart Re-positioned in Chart Window

This concludes the current lesson.

In addition to the Data Flow Diagram (DFD) chart type, EasyCASE Plus also offers several other chart types:

Chart Type	File Extension
Transformation Graph	.TRG
Structure Chart	.STC
Entity Relationship Diagram	.ERD
State Transition Diagram	.STD
Data Structure Diagram	.DSD
Data Model Diagram	.DMD
Entity Life History	.ELH
Logical Data Structure	.LDS

Creating a New Chart

For further practice, you may want to create the following ERD chart:

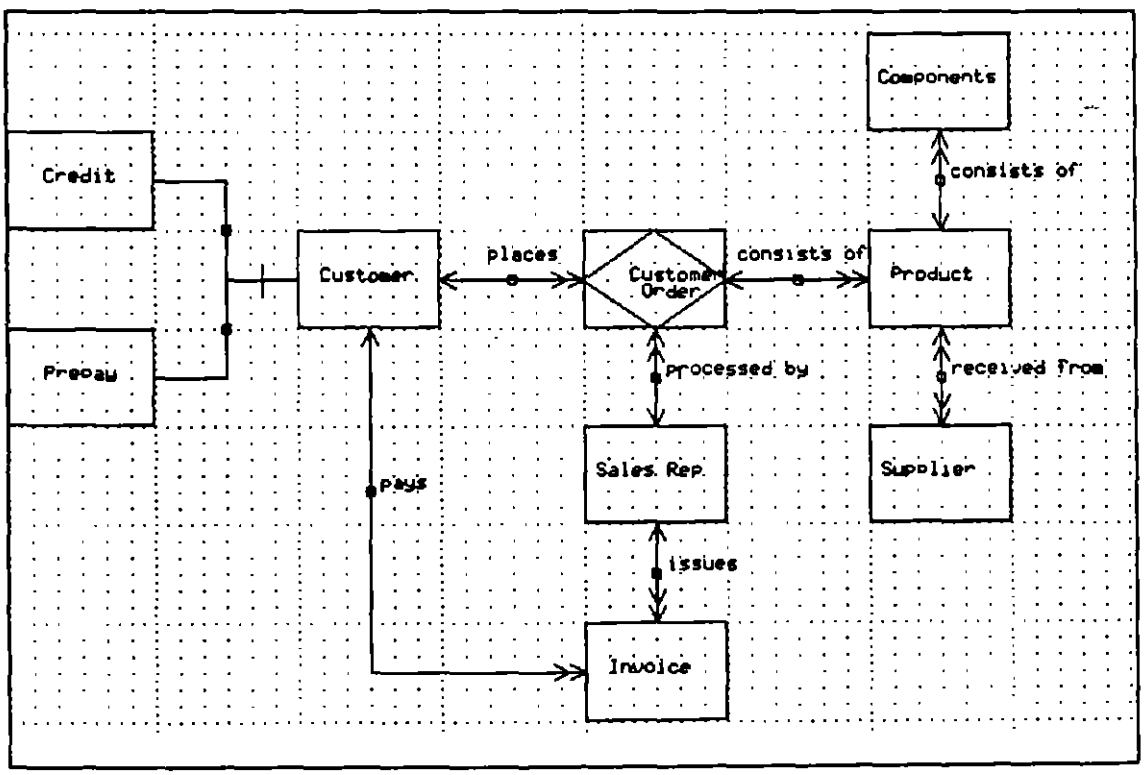


Figure 3.37 Alternative ERD Chart

You use the Change Type command from the File menu to access a different chart type as follows:

1. Select the 'Change Type' option from the File menu. If necessary, press the ESC key to bypass the 'Save chart?' dialog box if it appears.

   A listing of the supported chart types and symbol sets is displayed in the form of a pop-up list box, as shown in Figure 3.38.

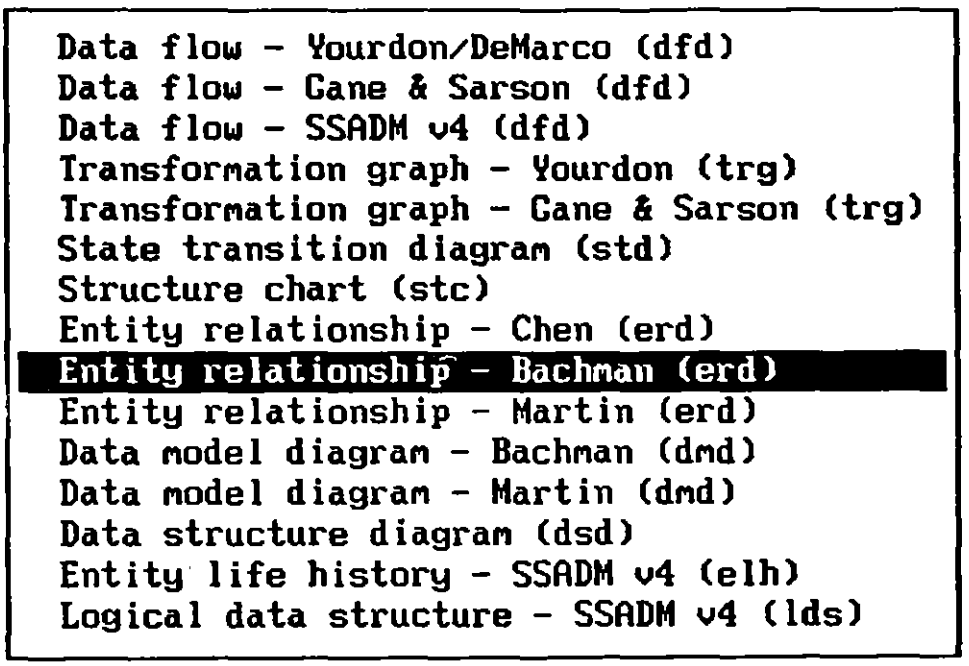


Figure 3.38 Listing of Supported Chart Types and Symbol Sets

2. Click on the 'Entity Relationship Diagram-Bachman (ERD)' entry to select it.

3. The EasyCASE Plus chart window screen is cleared and the Title Block and screen information updated for the new chart "UNTITLED.ERD".

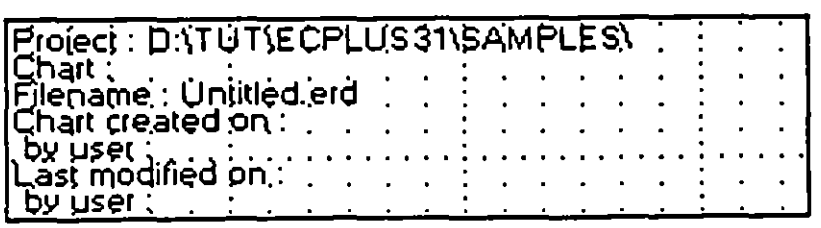


Figure 3.39 Changed Title Block

4. Notice the changed object type icons displayed to the left of the chart window. These represent available chart object types for an ERD. The entries in the Objects menu are similarly updated.

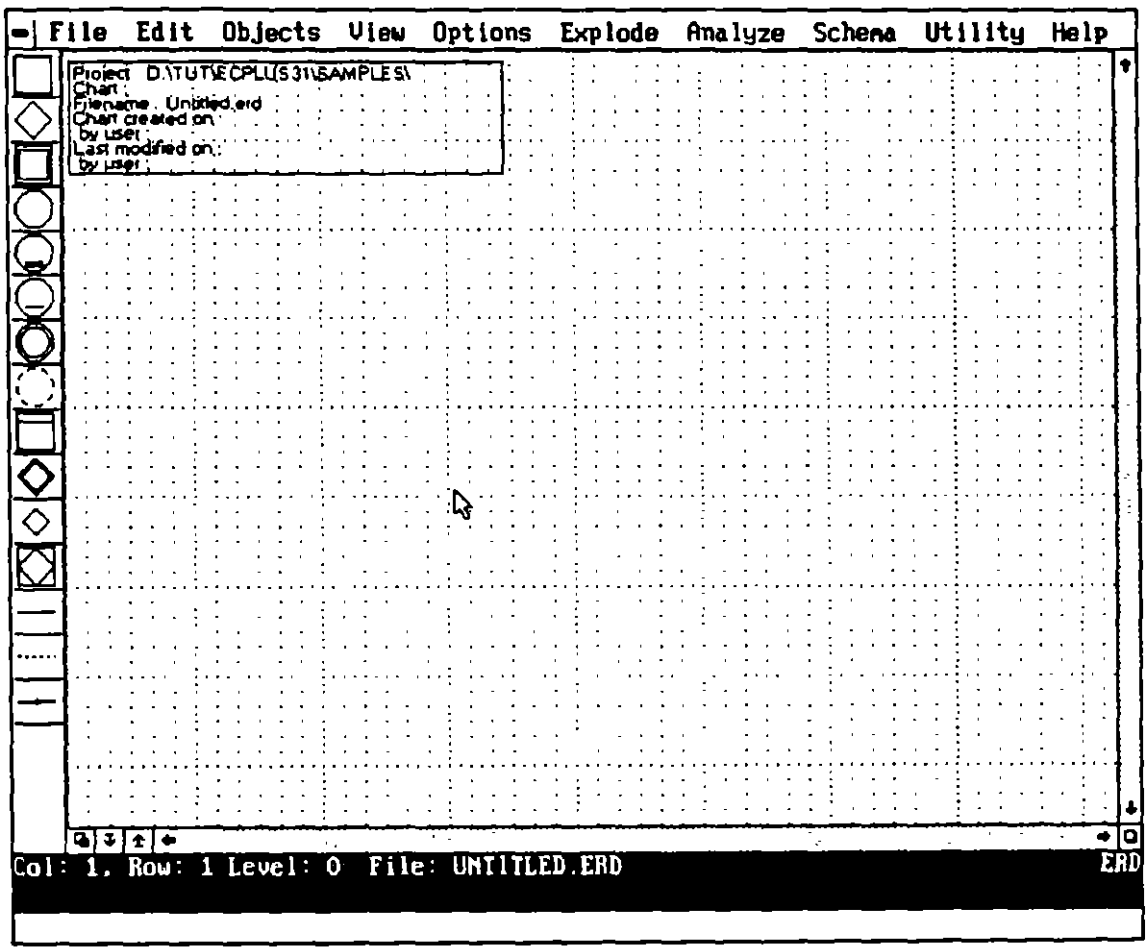


Figure 3.40 Bachman ERD Objects Icons

If necessary, refer to the procedures in Lesson 2 and this lesson to create a chart based on the sample that you have chosen.

For More Information

See "Selecting, Placing & Manipulating Chart Objects" in Chapter 5 of the EasyCASE Plus Chart Editor User's Guide.

LESSON 4
OTHER CHART FILE OPERATIONS

Overview

This lesson covers:

- Saving a selected group of objects to a new chart
- Merging a saved chart into a currently loaded chart
- Deleting a chart

For this lesson, you will use the TUTOR1.DFD chart you created in Lesson 2.

This lesson should take you approximately 15 minutes to complete.

Saving a Group of Chart Objects

Rather than always saving the complete chart, you may only want to save a selected part of a chart. You can do this with the Save Block option in the File menu.

To save the Order Processing System, Supplier and Payroll symbols area of the chart:

1. Select the Order Processing System, Supplier and Payroll symbols with their associated connections as a group, following the procedure provided in Lesson 1, as shown in Figure 4.1.

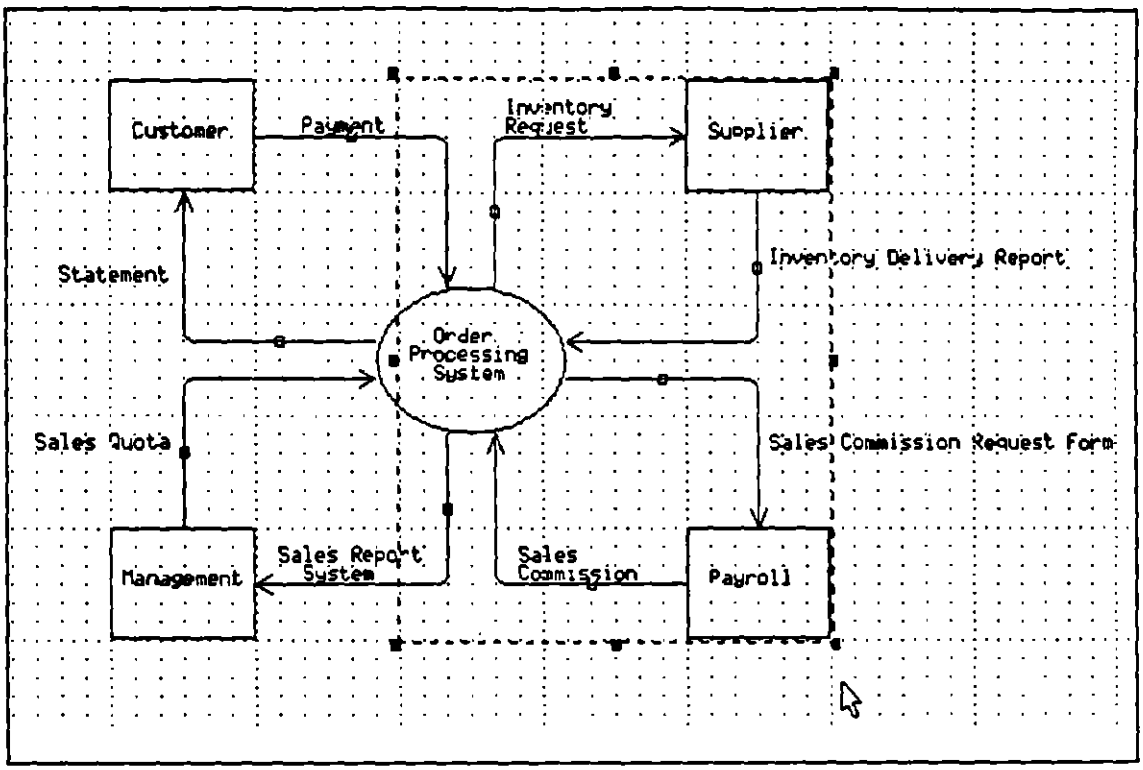


Figure 4.1 Selected a Group of Chart Objects

2. Select the 'Save Block' option from the File menu, or press CTRL+B.

3. A dialog box pops up prompting you for the filename to use for saving the selected block, as shown in Figure 4.2. The default filename is FRAGMENT.DFD.

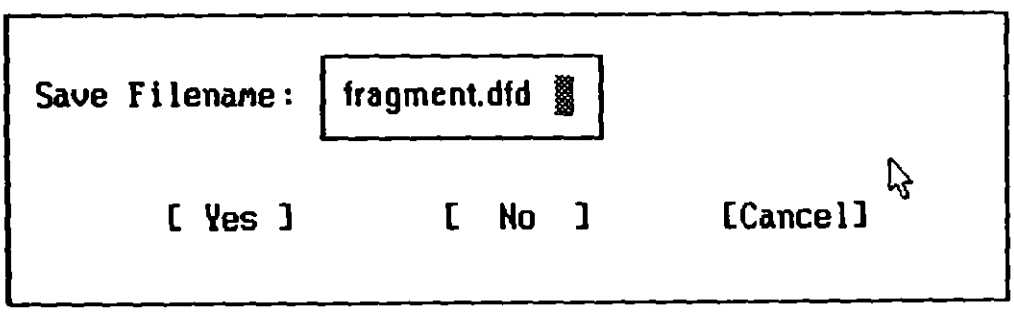


Figure 4.2 Prompt for Chart Filename to Save Selected Object Group

4. Click on the 'Yes' option to save the selected chart object group to a new chart using this default filename.

5. A second dialog box then pops up prompting you for the name to use for the new chart, as shown in Figure 4.3. The default chart name is "fragment".

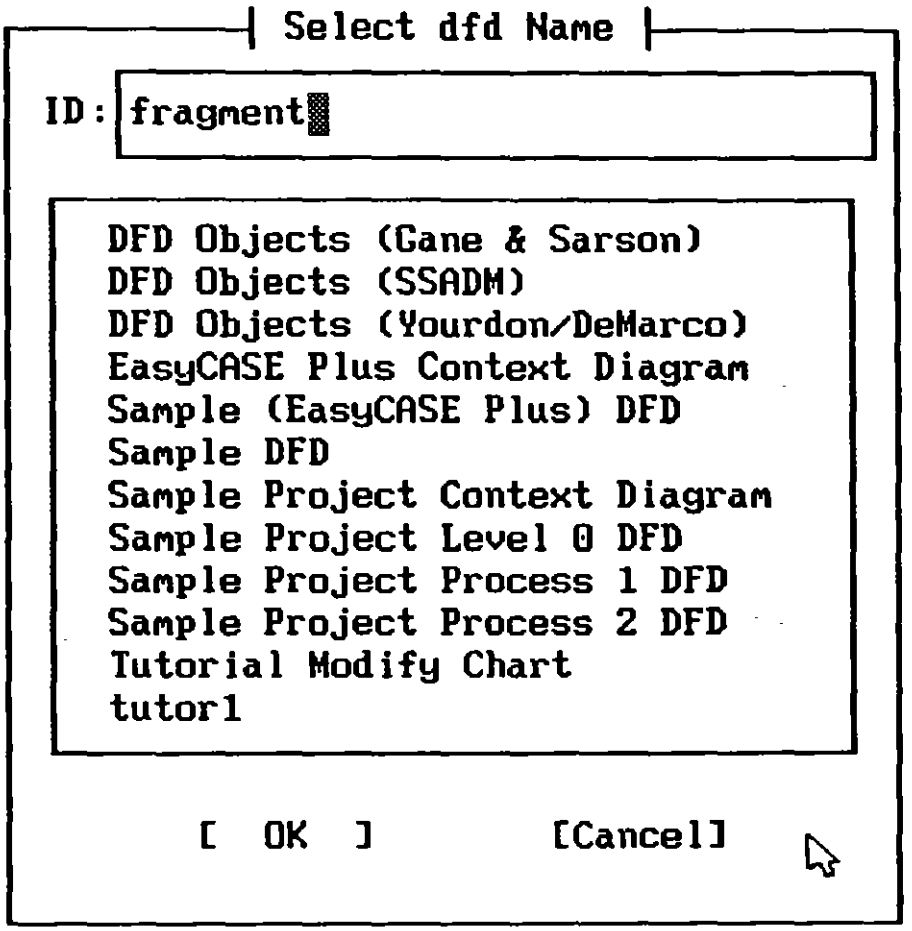


Figure 4.3 Prompt for Fragment Chart Name

6. Click on 'OK' to accept the default chart name presented.

7. The selected chart area is saved to file FRAGMENT.DFD in the current project directory.

The saved block of objects is still selected. Leave the block selected for the next (Merge Chart) operation.

Merging a Chart

You might want to merge a saved chart into the current chart. This procedure could save you time if you plan on using the same chart structure on more than one chart.

To merge the FRAGMENT chart you just saved into the current chart:

1. Choose the 'New chart' command from the File menu.

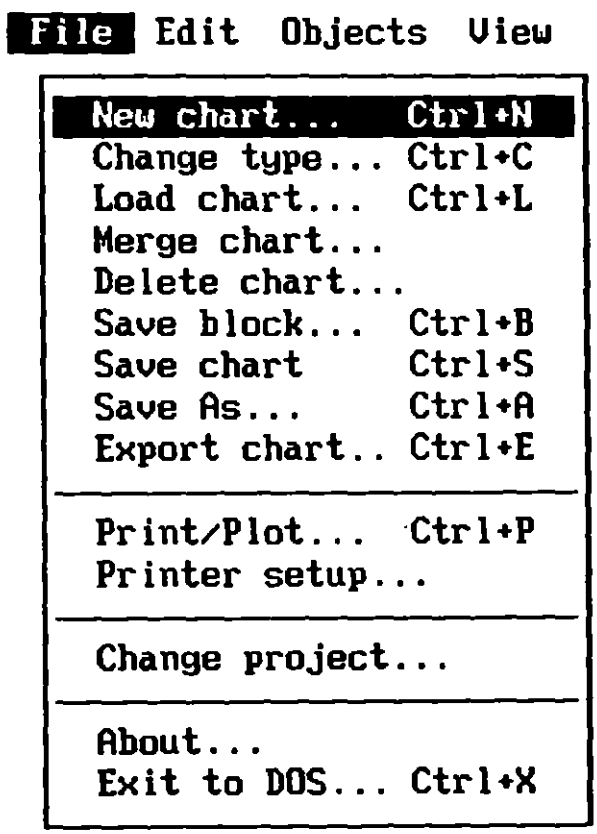


Figure 4.4 File Menu

2. An empty chart is displayed.

3. Select the 'Merge Chart' option from the File menu.

4. A listing of the DFD charts available in the current project directory is then displayed, as shown in Figure 4.5.

5. Click on the FRAGMENT chart entry (the chart object group that you previously saved).

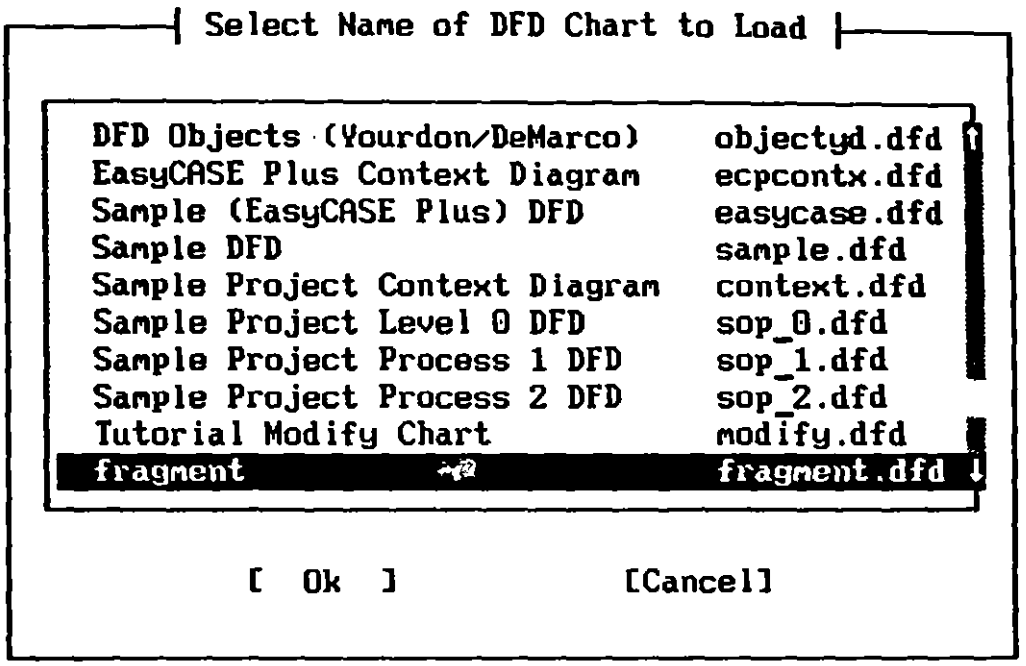


Figure 4.5 List of Charts Available to Merge

6. Click on the 'OK' option button.

7. A dialog box pops up to confirm merge of the FRAGMENT.DFD chart, as shown in Figure 4.6.

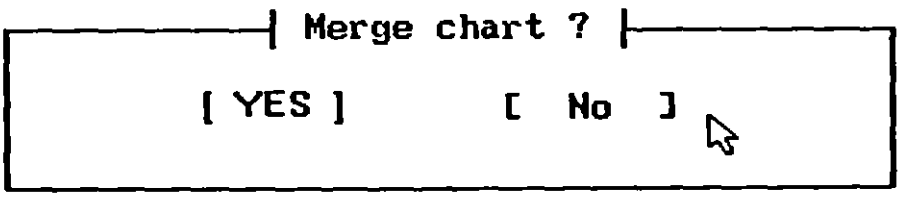


Figure 4.6 Prompt to Confirm Merge Chart

8. Click on the 'Yes' button.

9. Position the mouse pointer in the upper left corner of the chart window as shown in Figure 4.7.

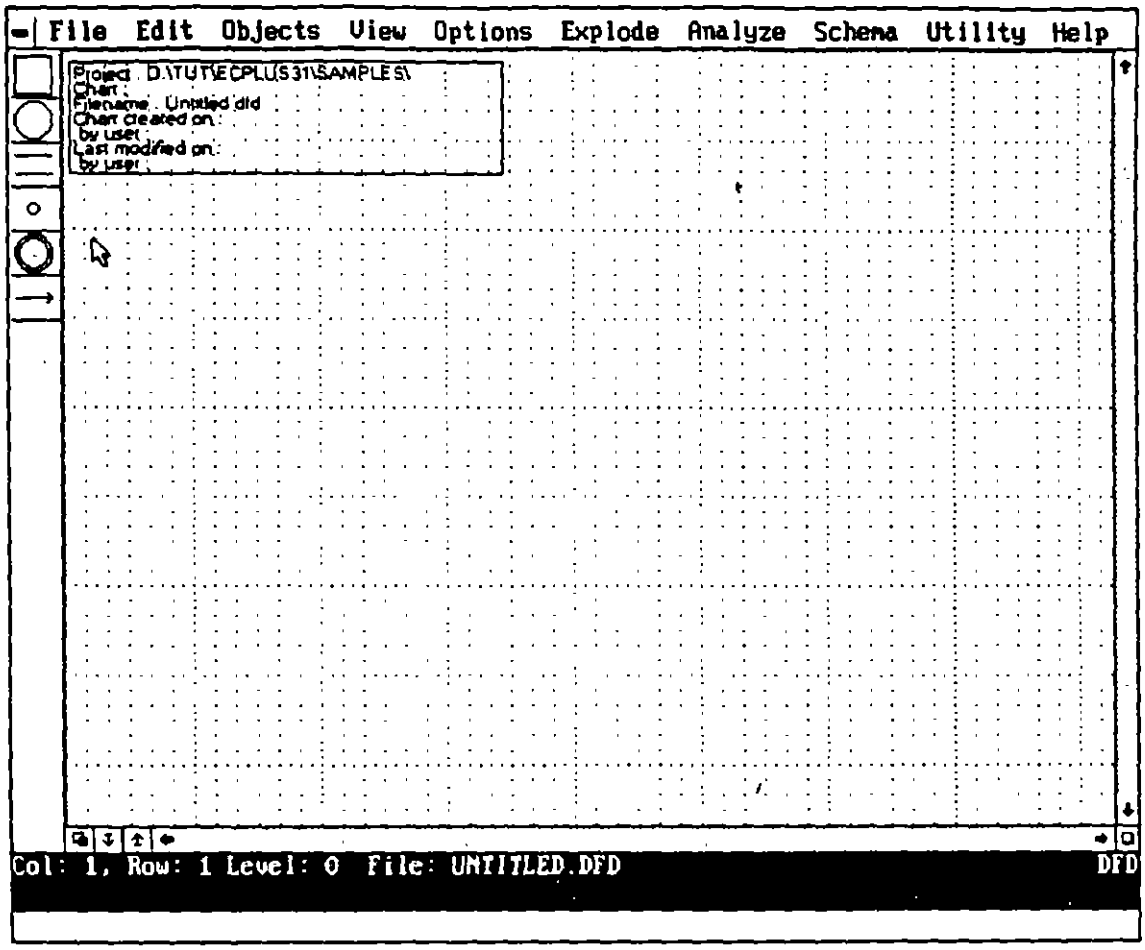


Figure 4.7 Mouse Pointer Positioned to Merge Chart

10. Click the left mouse button once.

    The FRAGMENT.DFD chart is then merged with the blank chart at the original position of the saved block, as shown in Figure 4.8.

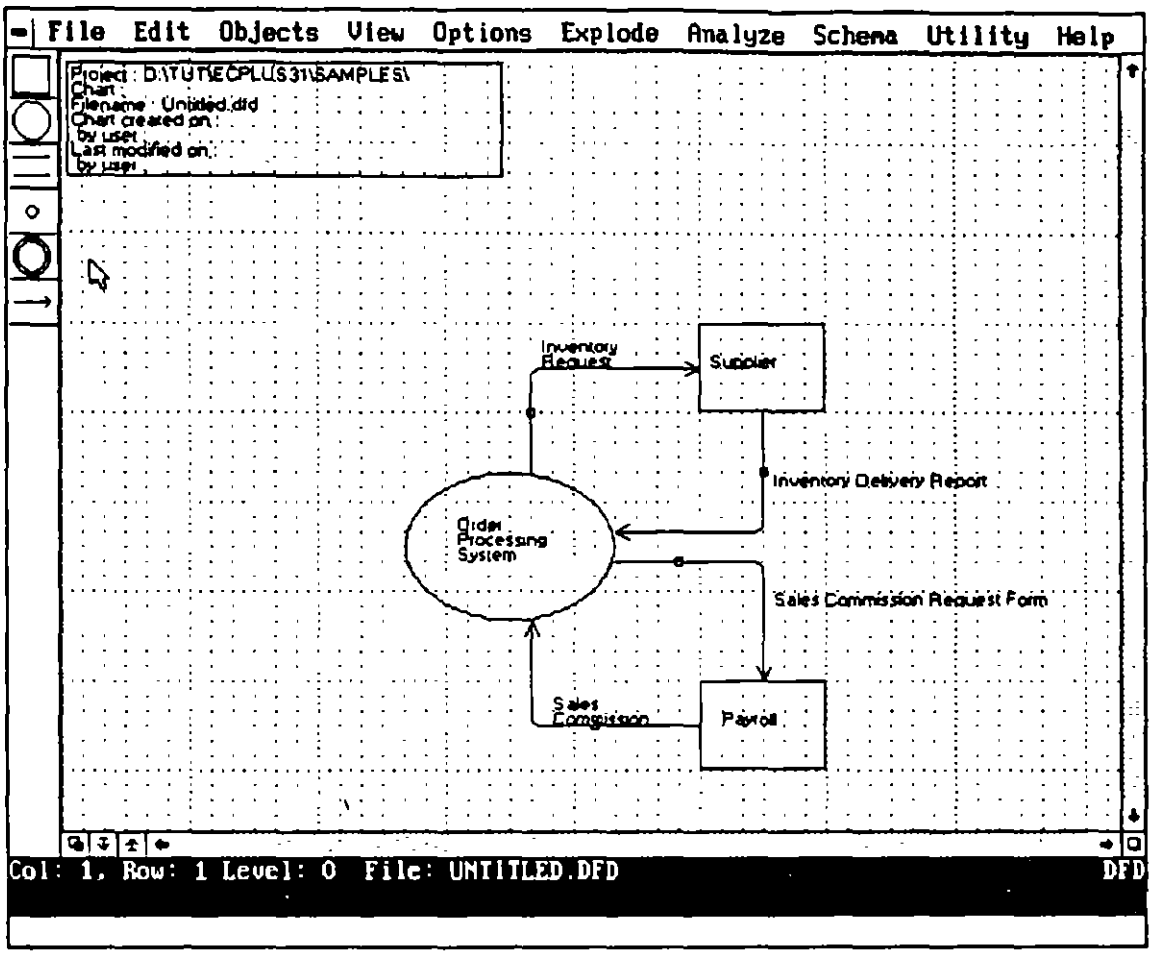


Figure 4.8 Merged Chart

Deleting a Chart

You no longer need the FRAGMENT.DFD chart containing the saved block of objects, so delete this chart, rather than using up storage space as follows:

1. Select the 'Delete Chart' option from the File menu (see Figure 4.4).

2. A listing of the available DFD charts in the current project directory is displayed.

3. Click on the FRAGMENT chart entry.

4. Click on 'OK.'

5. You are then prompted to verify that you want to delete the FRAGMENT.DFD chart, as shown in Figure 4.9.

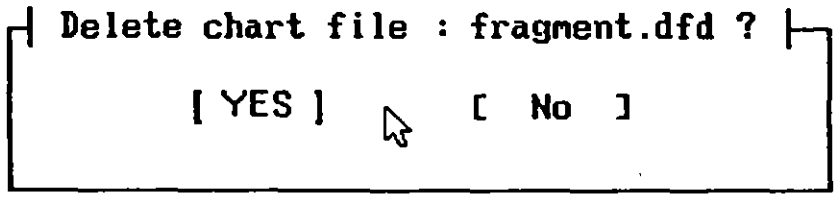


Figure 4.9 Prompt to Confirm Deletion of Selected Chart File

6. Click on 'Yes.'

7. You are then prompted to verify that you want to delete the data dictionary entry associated with the deleted chart, as shown in Figure 4.10.

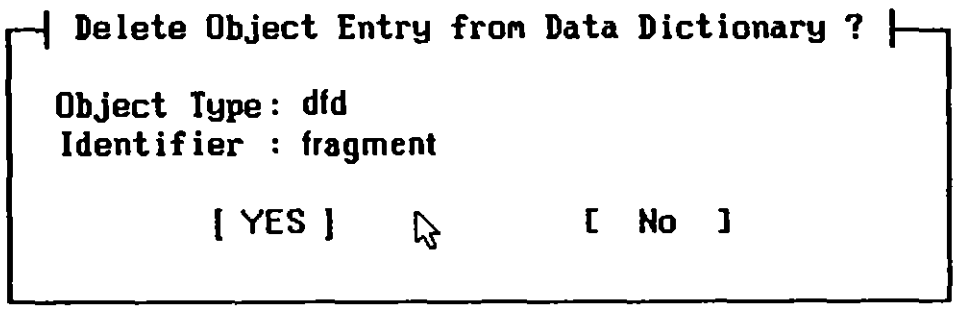


Figure 4.10 Prompt to Confirm Deletion of Deleted Chart's Data Dictionary Entry

8. Click on 'Yes.'

   The data dictionary entry for the FRAGMENT chart is marked for deletion, and the FRAGMENT.DFD chart is deleted.

NOTE: To permanently remove an identifier from the data dictionary, use the Pack command in the Data Dictionary and Reports Manager (DDRM) utility.

For More Information

See "Selecting, Placing & Manipulating Chart Objects" in the EasyCASE Plus Chart Editor User's Guide.

LESSON 5
DESCRIBING CHART OBJECTS TO THE DATA DICTIONARY

Overview

This lesson covers:

- Overview of chart object identification
- Identifying objects
- Data Dictionary Entry (DDE) screen access

You should now have a working knowledge of the skills covered in Lessons 1 through 4. The exercises in the following lessons build on these fundamental skills.

To save time, certain operations are mentioned but not the mechanics of how to achieve them, as these have been covered in prior lessons in detail.

For example, rather than directing you to load a chart and specifying the steps involved, you are simply directed to load the chart. Refer to the appropriate parts of the previous lessons if you need to refresh your memory.

This lesson should take you approximately 15 minutes to complete.

You will need to use the TUTOR1.DFD chart you created in Lesson 2 to complete the exercises in this lesson. If this chart is not currently loaded, you will need to load it now, as described in Lesson 3.

The Data Dictionary

The data dictionary stores information about chart objects and consists of "entries" or groups of attributes defining an object. Each entry in the data

dictionary has a unique identifier. However, multiple chart objects can be identified by a single entry in the data dictionary.

NOTE: In the EasyCASE Plus documentation you will see data dictionary entries referred to by the acronym DDE.

Some of the attributes that you specify in the data dictionary enable you to establish functional relationships or "links" to associated charts or text files, or to define data structures (such as records and elements). You will learn how to use these features in Lesson 6.

When you created objects in the Basic Skills Tutorial the objects were simply graphic symbols that you could modify or delete without affecting anything else in the diagram or project. They were not identified in the data dictionary and so did not have any potential for "real" functional relationships, and were not re-usable elsewhere in the project.

It is as though to be operable, each object has to have the equivalent of a Social Security number to uniquely identify it and that enables it to develop recognized associations with other data. The key word is "data". An object has to have a data component, its unique name in the data dictionary, before it can be a functional member of the chart.

By identifying chart objects and defining them in the data dictionary, you begin to tap the full potential of EasyCASE Plus.

Identifying Chart Objects

Throughout the Basic Skills Tutorial you created and modified objects that were not identified in the Data Dictionary. This is the main method of developing charts. When you have completed a chart, you can then go back and "manually" identify each object. This method is good for quickly prototyping charts, without having to be concerned with the underlying functional aspects. You can also manually identify objects immediately after you place them.

Manually Identifying Chart Objects

Having created the TUTOR1.DFD chart without identifying any of the objects (symbols and connections), you can now identify them as follows:

1. Select the data process symbol labeled "Order Processing System".

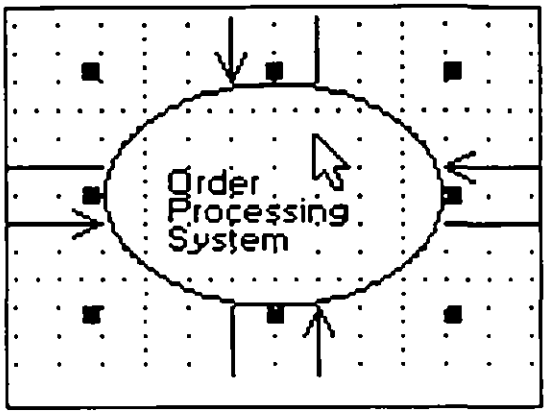


Figure 5.1 Selected Data Process Symbol

2. Select the 'Identify Object' option from the Edit menu, or press ALT+I.

3. The identifier dialog box for the symbol is displayed, with the cursor located in the text entry box at the top of the dialog box, as shown in Figure 5.2.

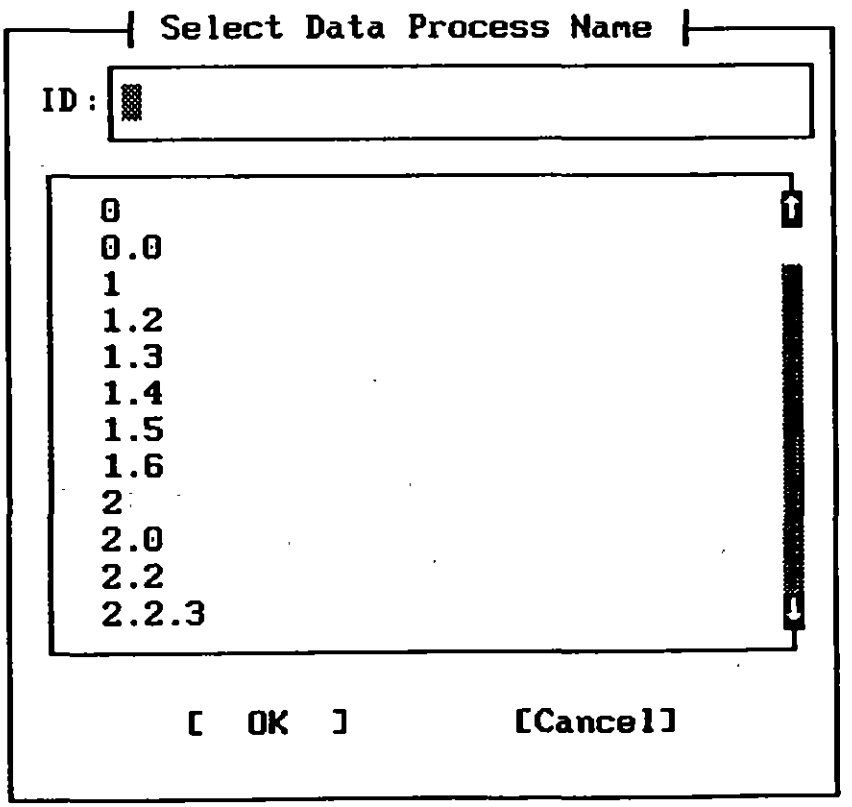


Figure 5.2 Identifier Dialog Box

Any identifiers for any other Data Processes already defined in the data dictionary for the Sample project are listed below the text entry box.

NOTE: If the symbol was already identified, the identifier would appear in the text entry box and highlighted in the list.

4. Type "Order Processing System" in the text entry box, as shown in Figure 5.3. This will be the symbol's identifier.

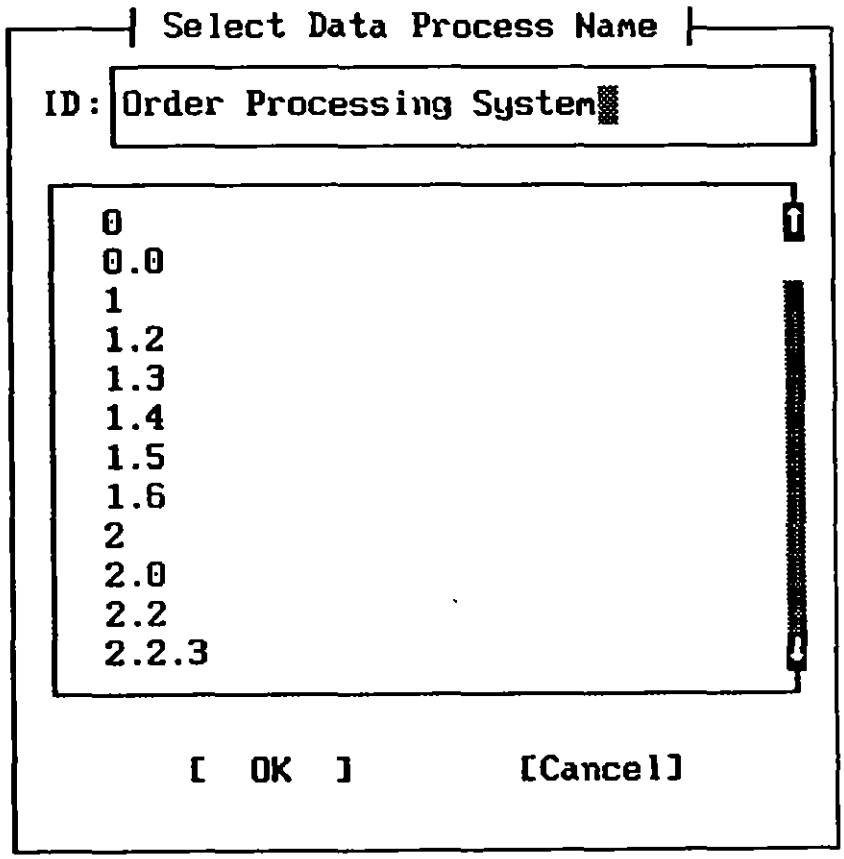


Figure 5.3 Symbol Identifier in Text Entry Box

5. Click on the 'OK' button.

6. The symbol's Data Dictionary Entry data entry screen is displayed as shown in Figure 5.4. The top two lines of the dialog box show the name (Order Processing System) and type (Data Process) of the selected symbol you just identified.

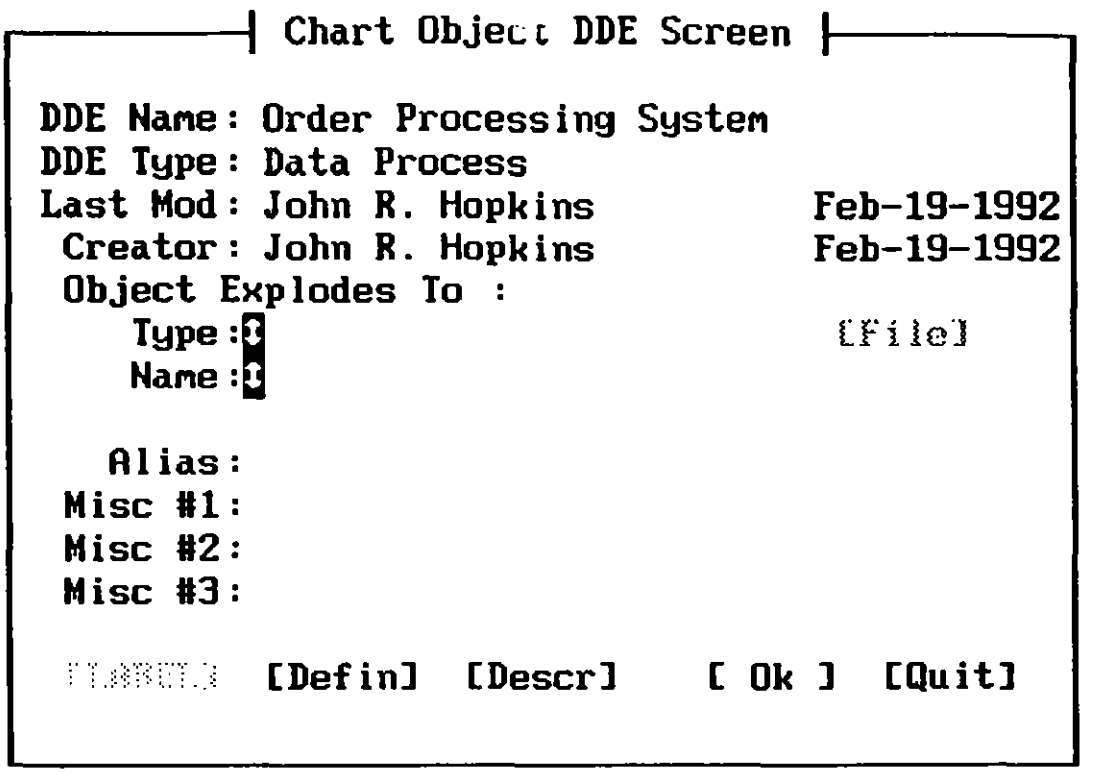


Figure 5.4 Data Dictionary Entry Screen

Notice the current attributes defined for the chosen object:

- The Name field shows the identifier for the symbol (cannot be edited).
- The Type field indicates the type of the symbol (cannot be edited).
- The Last Mod (modified by) attribute is the name of the person who last modified the Data Dictionary Entry for this symbol (cannot be edited) and the date the modification was made.
- The Creator (created by) attribute is the name of the person who created the DDE for this symbol (cannot be edited) and the date of creation.
- The [Label] button allows access to the existing label for the object. Although you can edit the label, do not do so at this time. It is the same label as the one you entered by choosing the 'Label Object' option from the Edit menu after you placed the object on the chart.

NOTE: Do not be concerned with the other options displayed at this point. We will address those in Lesson 6.

7. Click on the 'OK' button or press F3 to return to the chart.

   Both the symbol and its label are selected.

8. Press the ENTER key to leave the label positioned at its original position. You have just defined the selected chart object in the data dictionary.

9. Repeat steps 1 to 5 for the Customer, Management, Supplier and Payroll external entity symbols, entering identifiers of "Customer", "Management", "Supplier", and "Payroll" for each respectively.

Automatically Identifying Chart Objects

If you prefer, rather than creating your objects then going back and identifying them, you can choose to have EasyCASE Plus automatically prompt you to identify objects when you create them. Although this adds some time to the chart creation process, you do not have to return to the chart later to finish defining the objects in the data dictionary.

As a convenience, you can set the 'Auto Identify' option On. In this case, EasyCASE Plus will prompt you to identify each object as it is placed on the chart. This saves you the step of invoking the Identify command from the Edit menu after placing an object on the chart.

To set the 'Auto Identify' option to On:

1. Select the Options menu.

2. Select the Preferences entry.

   A dialog box will pop up listing a number of available options, as shown in Figure 5.5.

3. Click on the 'Auto Identify' option to set it to On. When it is On, an 'X' will appear to the left of this option's text.

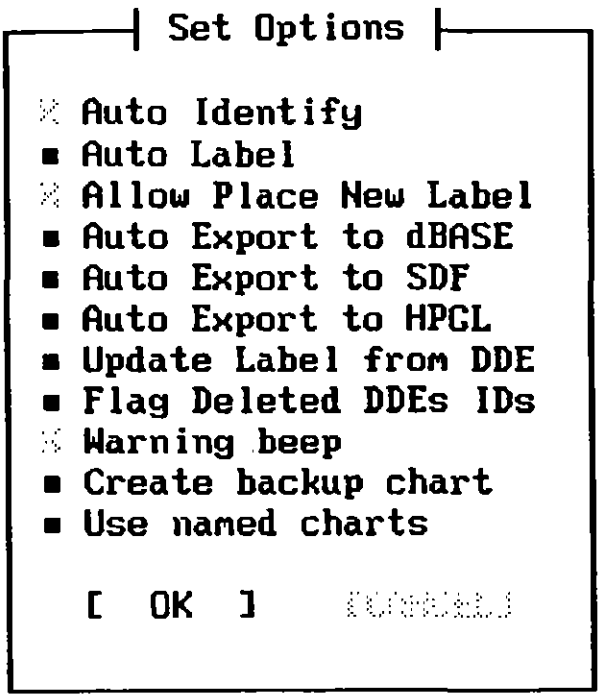


Figure 5.5 Preferences Dialog Box

4. Click on the 'Cancel' button.

NOTE: There are a number of options in this dialog box that control the operation of the chart editor. These are described fully in the Chart Editor User's Guide.

Additional Information

You would use the same approach for identifying connections on any chart type, and for couples on structure charts.

NOTE: You do not strictly have to identify chart objects. If you do not however, you will be unable to link them to other charts, text files or data definitions as described in Lesson 6.

For More Information

See "Describing Chart Objects to the Data Dictionary" in the EasyCASE Plus Chart Editor User's Guide.

LESSON 6
LINKING CHART OBJECTS VIA THE DATA DICTIONARY

Overview

This lesson covers:

- The Data Dictionary Entry (DDE) screen
- Establishing a link to another chart
- Using the Explode commands to access the linked chart
- Establishing a link to a text file
- Using the Explode commands to edit the linked text file

Before beginning this lesson, you should be familiar with the material covered in Lesson 5, in which you simply identified several chart objects and briefly accessed the corresponding Data Dictionary Entries (DDEs). You did not actually add to, or modify, any of the DDE information.

You will need to use the TUTOR1.DFD chart you created in Lesson 2 (and updated in Lesson 5) to complete the exercises in this lesson. If this chart is not currently loaded, you will need to load it as described in Lesson 3.

This lesson should take you approximately 1 hour to complete.

In previous lessons you assigned the fundamental attributes for your Data Dictionary Entries. The objects you identified can now be used to establish relationships: functional links to other charts or text files to create a hierarchy for your project files as described in the following sections.

Linking a Chart Object to Another Chart

In this section, you will use the Data Dictionary Entry for the "Order Processing System" Data Process symbol as an example and link it to another chart.

Access the DDE for the "Order Processing System" data process symbol as follows:

1. Select the "Order Processing System" data process symbol. It will be highlighted.

2. Select the 'Identify Object' option from the Edit menu, or press ALT+I.

3. The identifier list box is displayed with the identifier "Order Processing System" in the text entry box, as shown in Figure 6.1. A highlight bar appears over that entry in the list indicating that the identifier is assigned to the selected chart object.

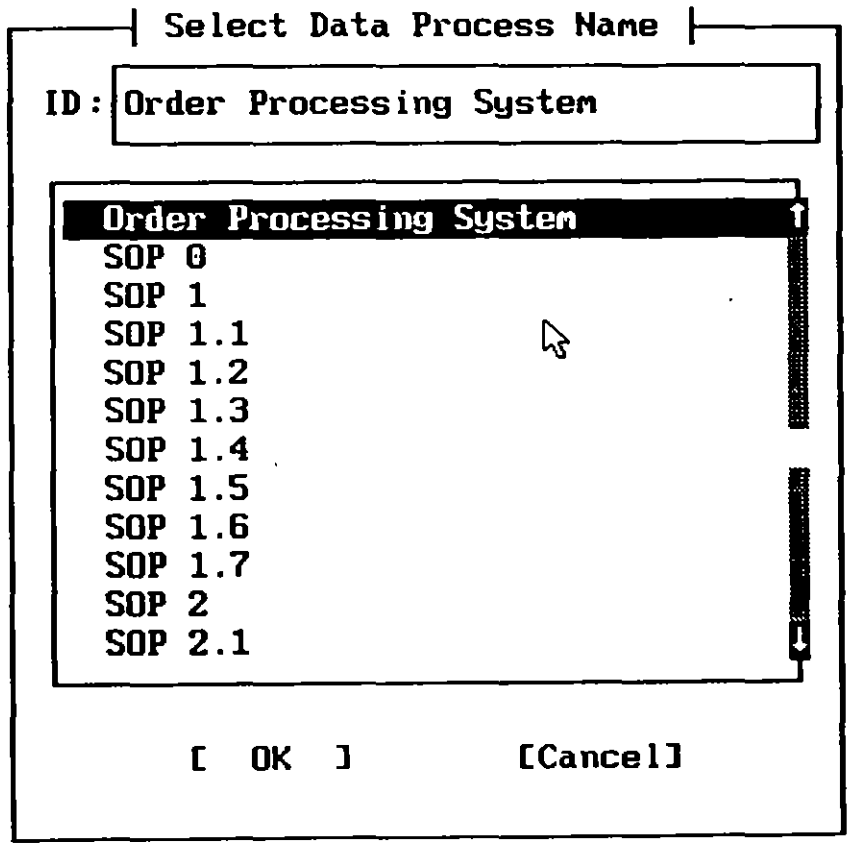


Figure 6.1 Identifier List Box with Object Identifier Highlighted

4. Click on 'OK' to retain that identifier.

5. The DDE screen is displayed for the selected "Order Processing System" Data Process symbol, as shown in Figure 6.2.

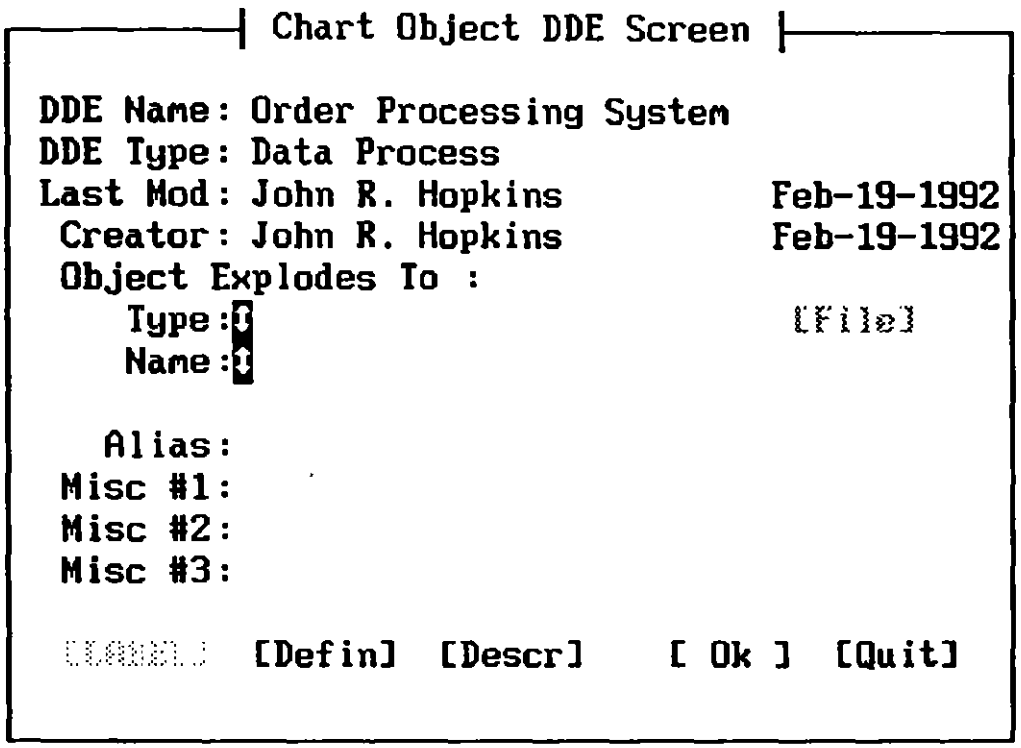


Figure 6.2 DDE Screen Displayed for Selected Object

Note that the Explodes To Type and Name attributes are blank indicating that the selected chart object has not (yet) been linked to any other child object.

In this section, you will create a link from the "Order Processing System" data process symbol to the MODIFY.DFD chart (another DFD chart in the SAMPLES project directory).

The link associates a specific chart object with another chart. You then use the 'Explode Down' function to load the linked chart.

To specify the name and type of the chart to link to:

6. Click on the up/down arrow icon next to Object Explodes To - Type field.

7. A dialog box, listing the available child chart types, data definition types, and text file types is displayed, as shown in Figure 6.3.

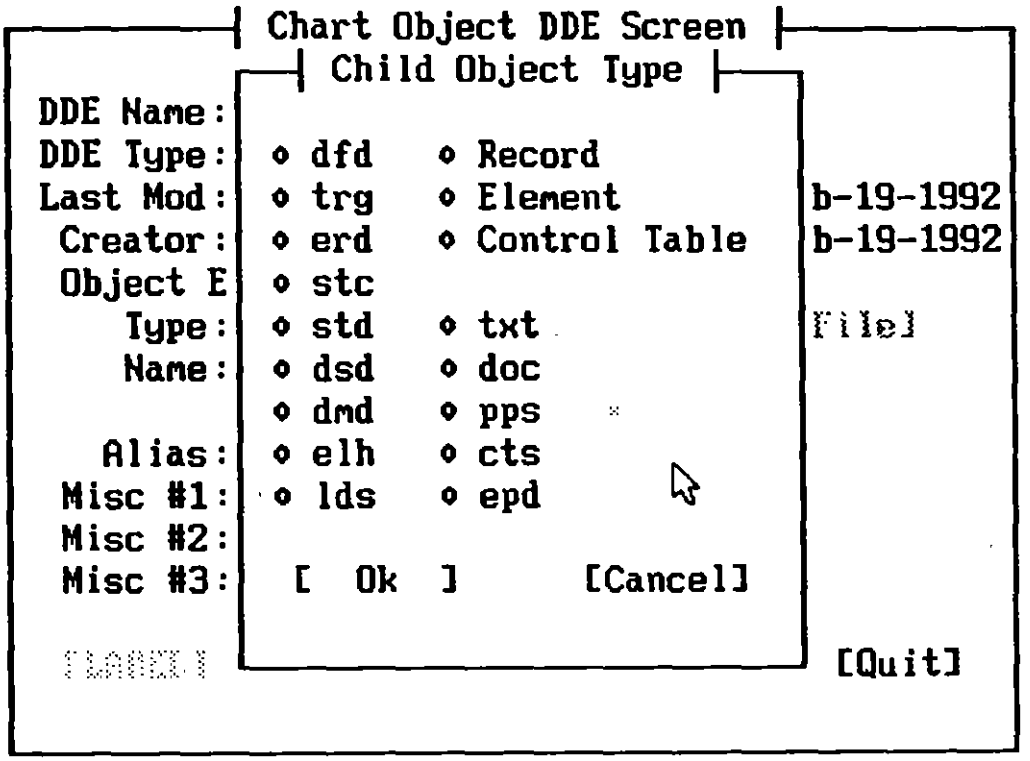


Figure 6.3 Dialog Box Listing Available Child Object Types

8. Click on the 'dfd' button to select it.

9. Click on the 'OK' option button.

   The text 'dfd' appears immediately to the right of the Object Explodes To - Type field indicating that a link is being established between the selected data process symbol on the current DFD to another data flow diagram.

   Next you will specify which particular DFD, by name, you want to link to.

10. Click on the up/down arrow icon next to the Object Explodes To - Name field.

    A dialog box appears listing the names of all existing, available data flow diagrams so far described in the data dictionary, as shown in Figure 6.4.

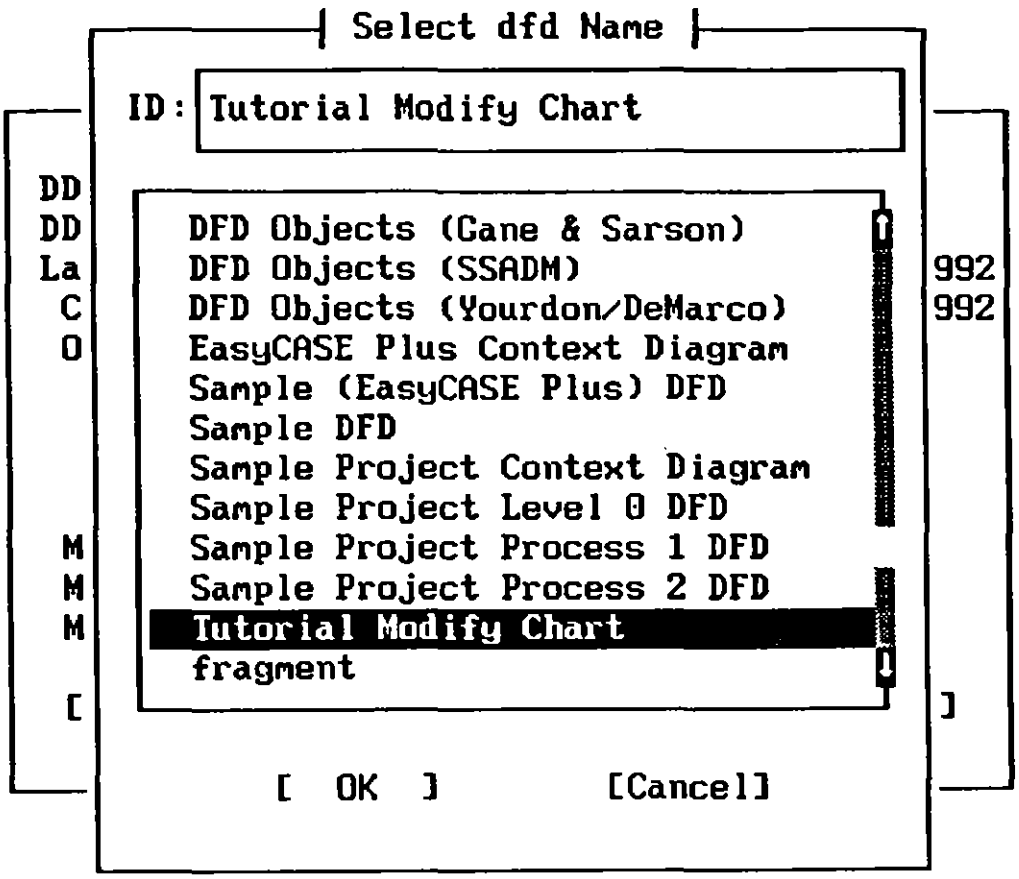


Figure 6.4 Dialog Box Listing Available DFD Charts

11. Select the 'Tutorial Modify Chart' entry by clicking on it.

12. Click on 'OK.'

13. The Data Dictionary Entry screen for the Order Processing System is redisplayed as shown in Figure 6.5. Notice the entries in the Object Explodes To - Type and Explodes To - Name fields are those you just chose from the lists. In this example, the DDE screen shows that the data process named "Order Processing System" has been linked to another data flow diagram (dfd) named "Tutorial Modify Chart." In the next section you will traverse the linkage and access the named 'child' chart.

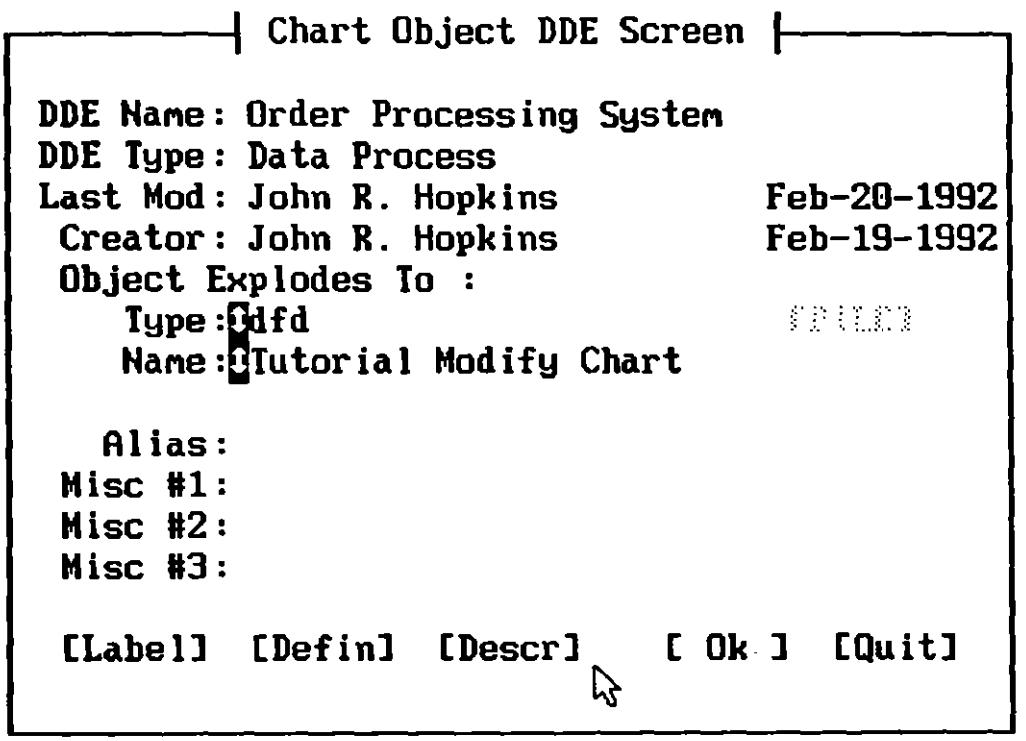


Figure 6.5 DDE Screen with Linked Chart Entry

14. Now click on the '[Descr]' button.

    A text entry box pops up, overlaying the DDE screen, as shown in Figure 6.6. This text entry box allows you to add a short description about the selected data process.

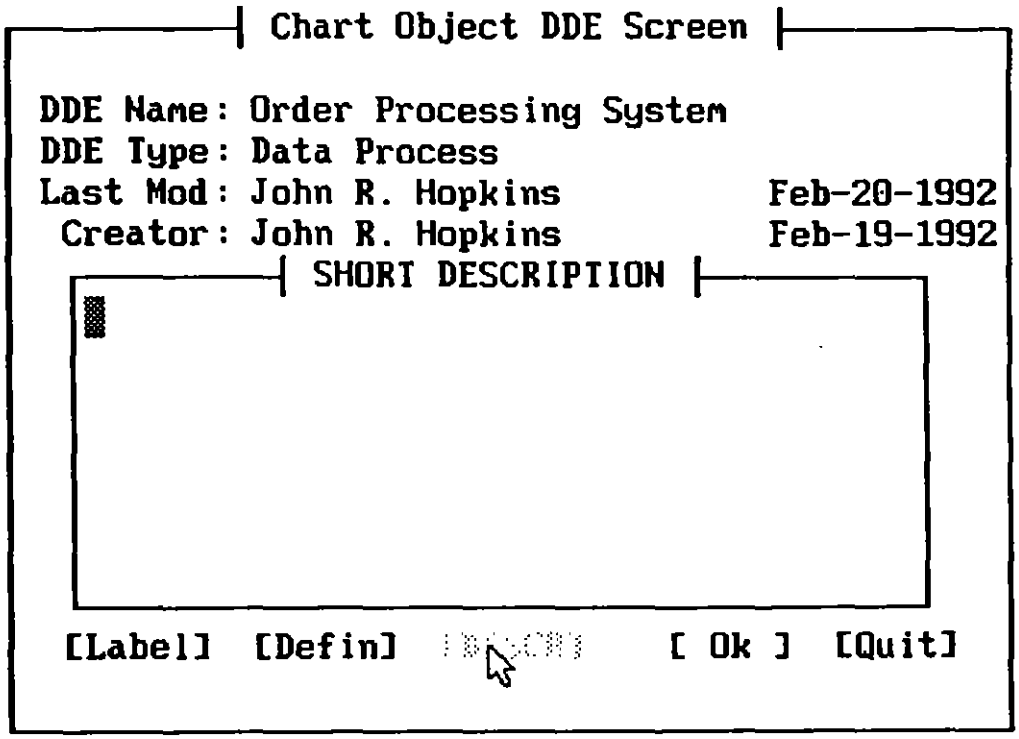


Figure 6.6 Short Description Field Text Entry Window

15. Type the following short description for the Order Processing System entry: "Process representing the entire Order Processing System".

16. Press the F3 key to complete entry of the short description.

17. Click on 'OK' to return to the chart.

    The object label is highlighted along with the selected symbol.

18. Click the right mouse button, or press the ENTER key, to accept the current position for the symbol label.

When you accessed the DDE screen in this example, you defined a link between the selected chart object and another chart via the data dictionary (chart of type DFD named 'Tutorial Modify Chart'). You can now 'explode' down from the selected data process to the designated chart, as described in the next section.

Exploding To and From a Linked Chart

Once you have established a link to a chart, it is a simple matter to "explode" down to the Child chart, automatically loading it into the chart window ready to be worked on. To do this, you select the object linked to the chart, then use the 'Down' option from the Explode menu.

To explode to the linked chart from the Order Processing System data process:

1. Select the data process symbol labeled "Order Processing System" by clicking on it. The symbol is highlighted.

2. Select the 'Down a Level' option from the Explode menu, or press CTRL + D, or click on the explode down icon located in the lower left corner of the chart window.

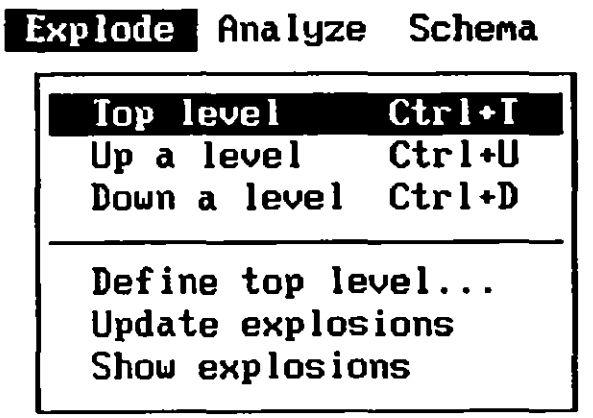


Figure 6.7 Explode Menu

If the current chart has changed since it was last saved, you are prompted to save the chart. To do so, click on the 'Yes' button.

3. The linked (child) chart (MODIFY.DFD) is loaded and displayed in the chart window ready to be worked on, as shown in Figure 6.8.

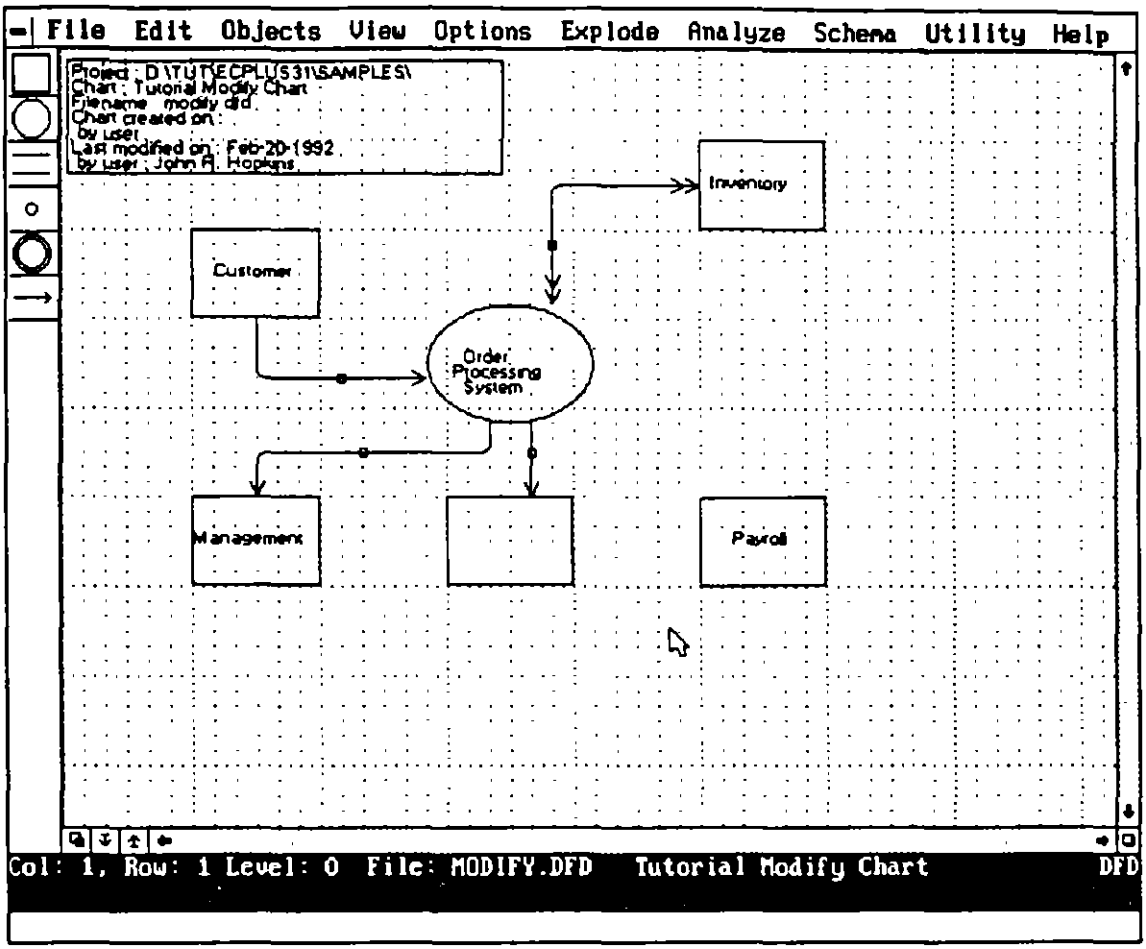


Figure 6.8 Child Chart Displayed in Chart Window

You can explode back to the parent chart (TUTOR1.DFD) by using the 'Up' option from the Explode menu, as follows:

1. Select the 'Up a Level' option from the Explode menu, or press CTRL + U, or click on the explode up icon located in the lower left corner of the chart window.

2. The parent chart (TUTOR1.DFD) is loaded and displayed in the chart window.

Only one chart can be loaded at a time. However, exploding from/to a chart is a quick and easy way to move between linked charts.

Linking to a New Chart

You have just defined a linkage from a existing chart object to another chart via the data dictionary. However, the chart that you wish to link to need not already exist. The new chart is created as you complete the link by attempting to access it.

To do this, you will use the 'Down a Level' option from the Explode menu.

1. Select the "Customer" external entity symbol by clicking on it.

2. Select the 'Down a Level' option from the Explode menu, or press CTRL + D, or click on the explode down icon located in the lower left corner of the chart window.

3. You will be asked if you wish to save the chart if it has been modified since it was last saved.

   Click on 'Yes' to save the chart.

4. A dialog box is displayed and lists the available child object types (charts, text files, data definitions), as shown in Figure 6.9.

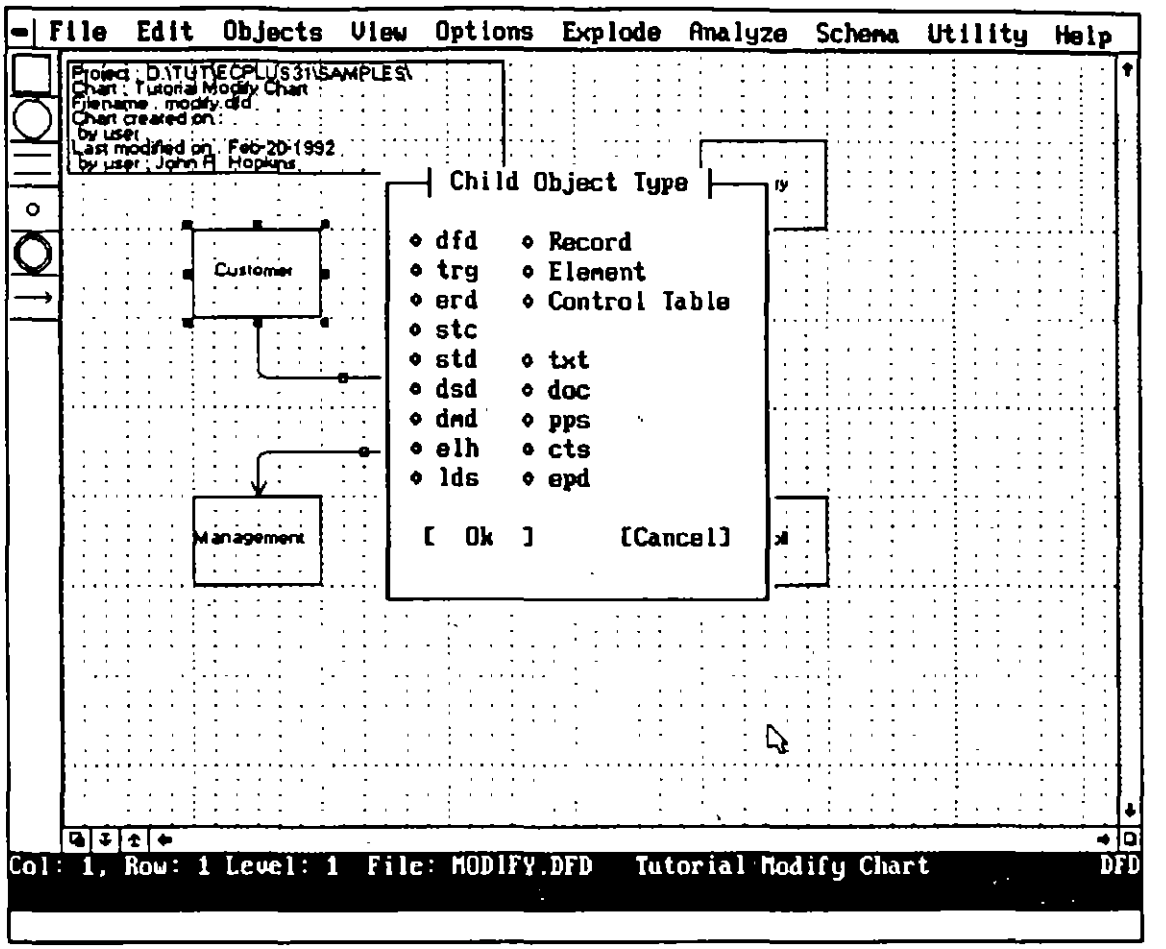


Figure 6.9 Available Child Object Types

5. Click on the 'erd' entry to choose an Entity Relationship Diagram as the child chart type you want to link to.

6. Click on 'OK.'

7. A dialog box is displayed containing the identifiers for ERD chart types in the current project. The text entry box at the top of the dialog box is empty and ready for text entry.

8. Type "Customer Operations" in the text entry box, as shown in Figure 6.10. This is the name of the ERD to link to.

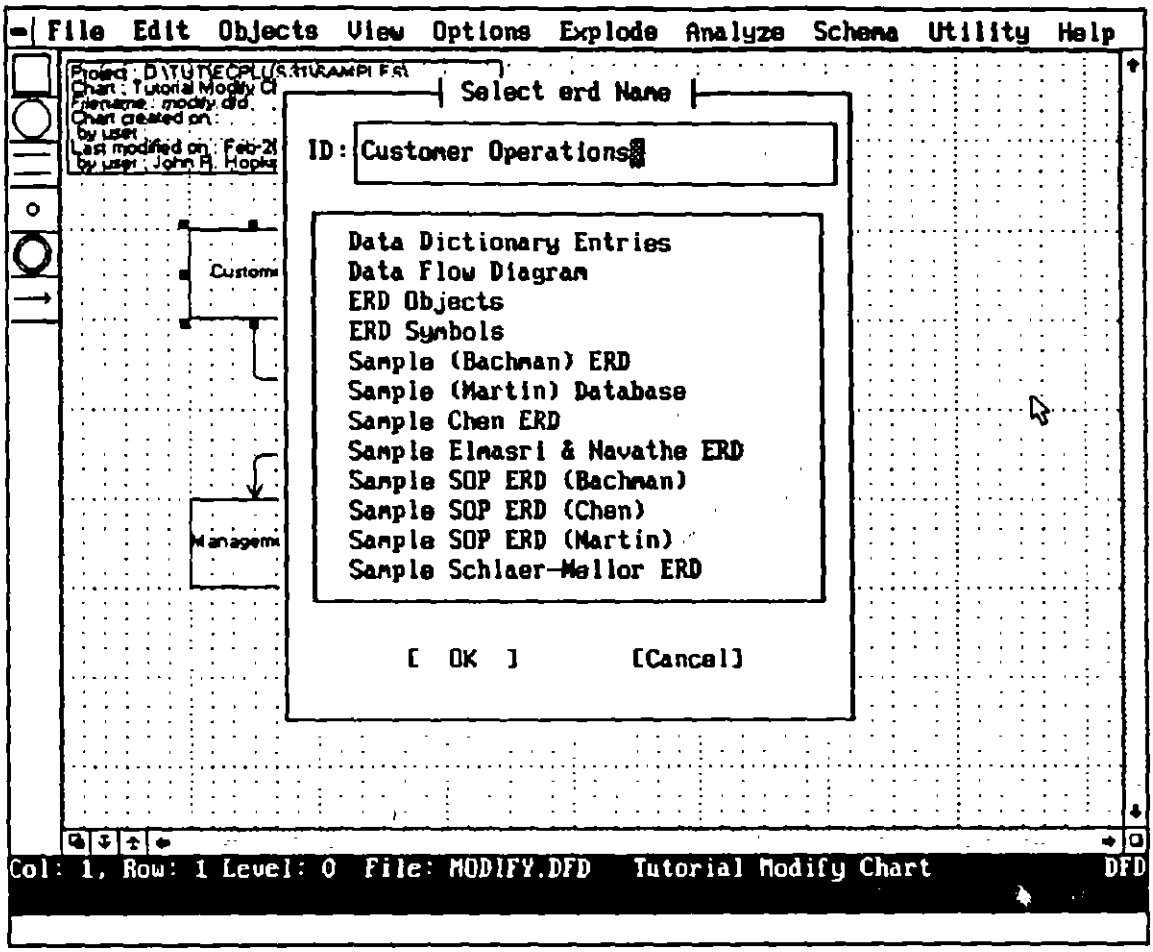


Figure 6.10 Child ERD Identifier in Text Entry Box

9. Click on 'OK.'

10. A dialog box then pops up and you are prompted to verify that you want to create the specified child chart entry in the data dictionary, as shown in Figure 6.11.

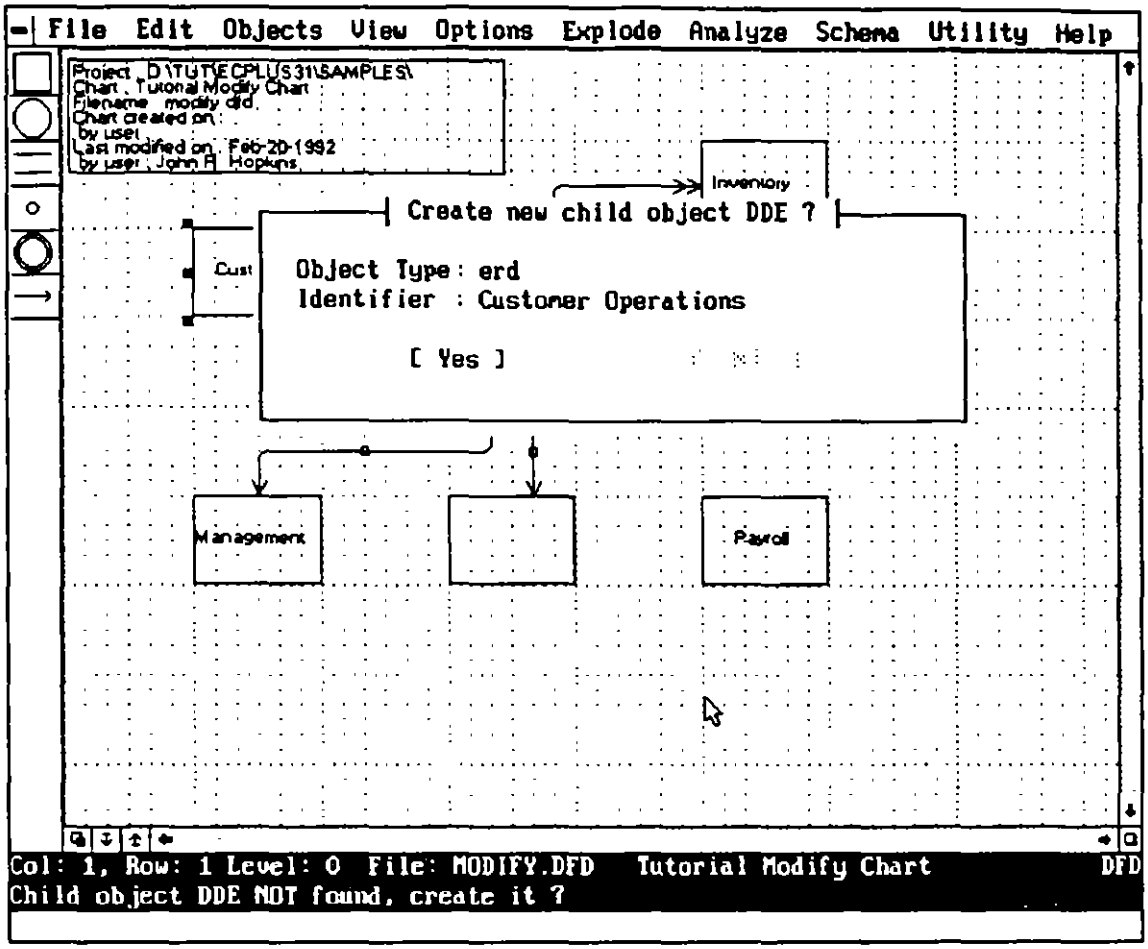


Figure 6.11 Prompt to Create Child Chart Entry in the Data Dictionary

11. Click on 'Yes.'

12. You are prompted to enter the filename to use for the ERD chart file, as shown in Figure 6.12.

13. Type "CUSTOPS.ERD" in the text entry box, to be used as the filename for the child ERD chart.

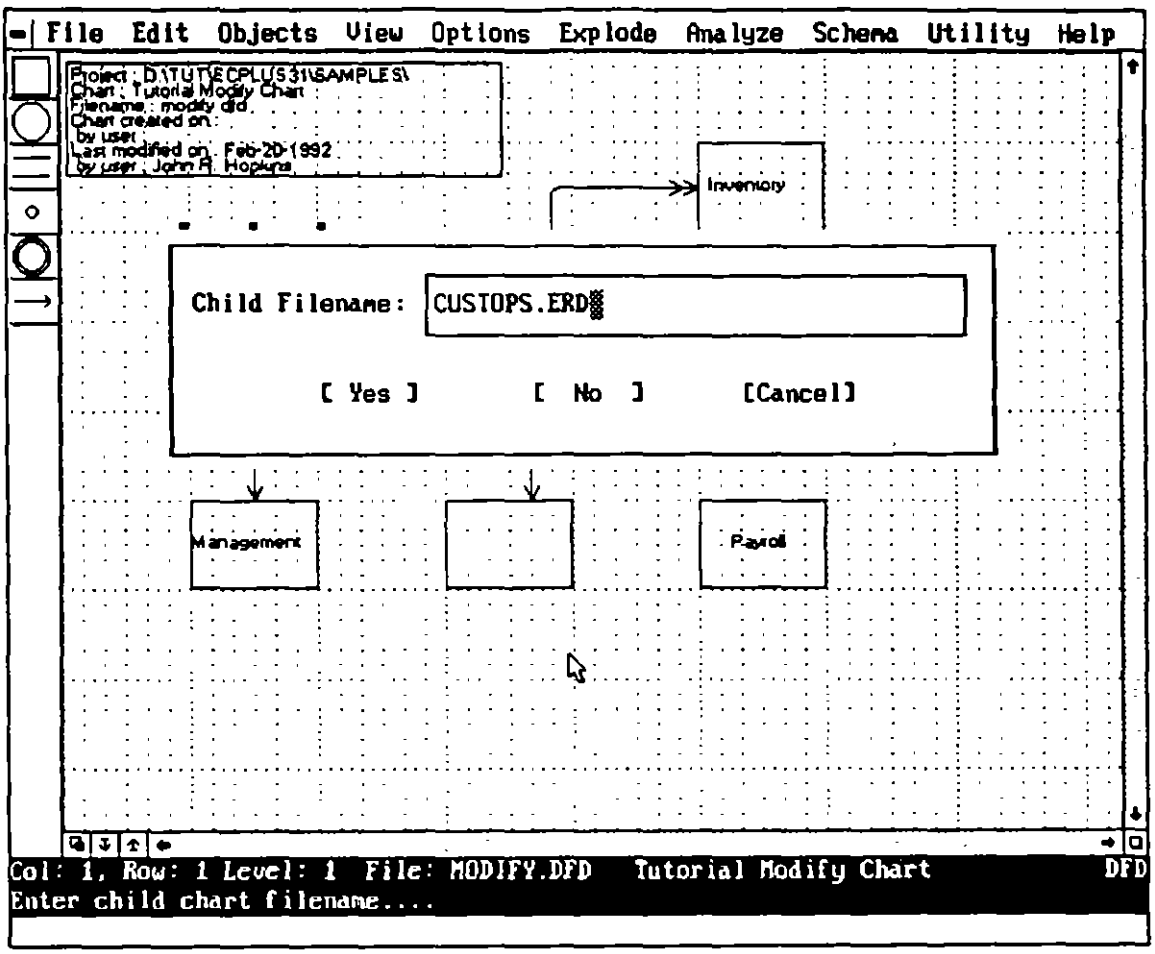


Figure 6.12 Text Entry Box for ERD Chart Filename Entry

14. Click on 'Yes.'

15. You are prompted to verify that you want to create the new ERD chart, as shown in Figure 6.13.

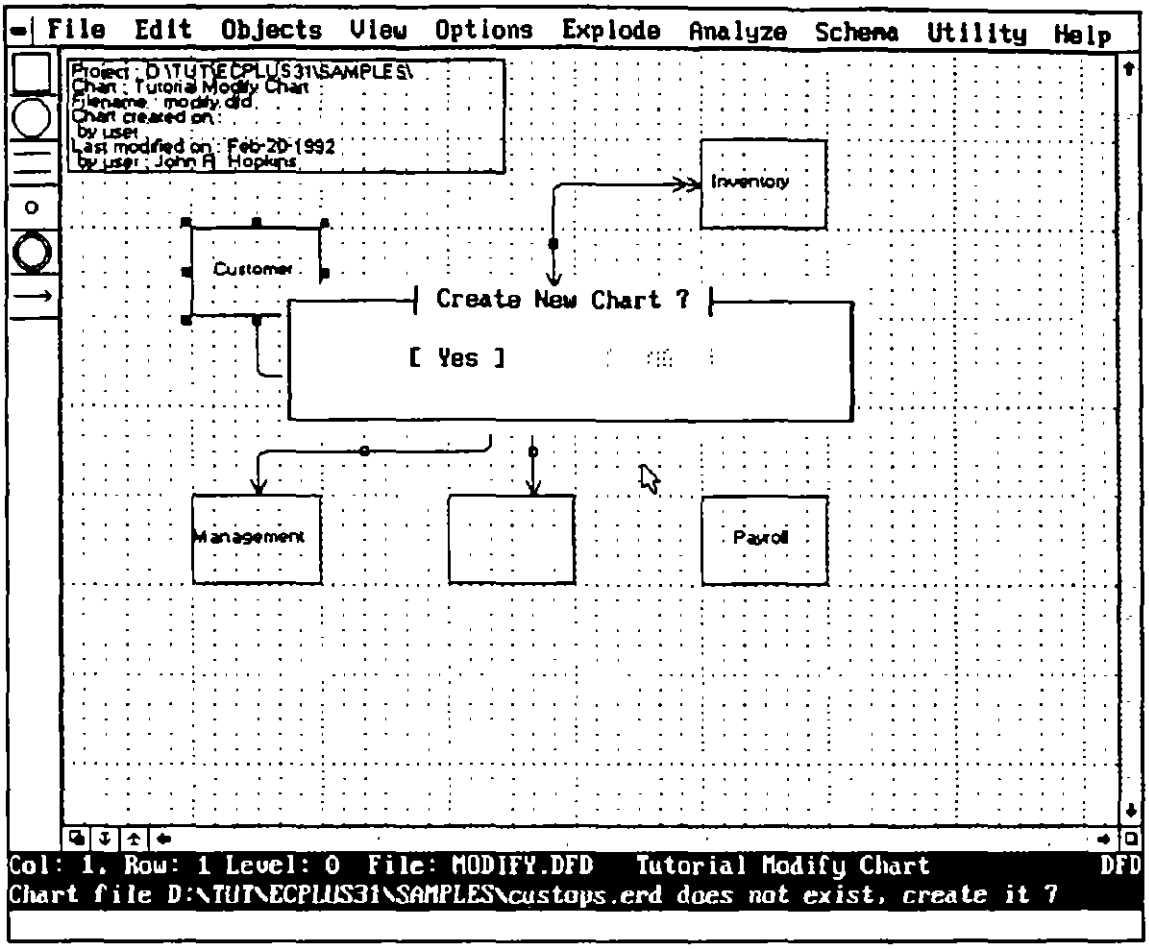


Figure 6.13 Prompt to Confirm Creation of New Chart

16. Click on 'Yes.'

17. The new CUSTOPS.ERD chart is then created, and a blank chart is displayed in the chart window, ready for you to work on, as shown in Figure 6.14.

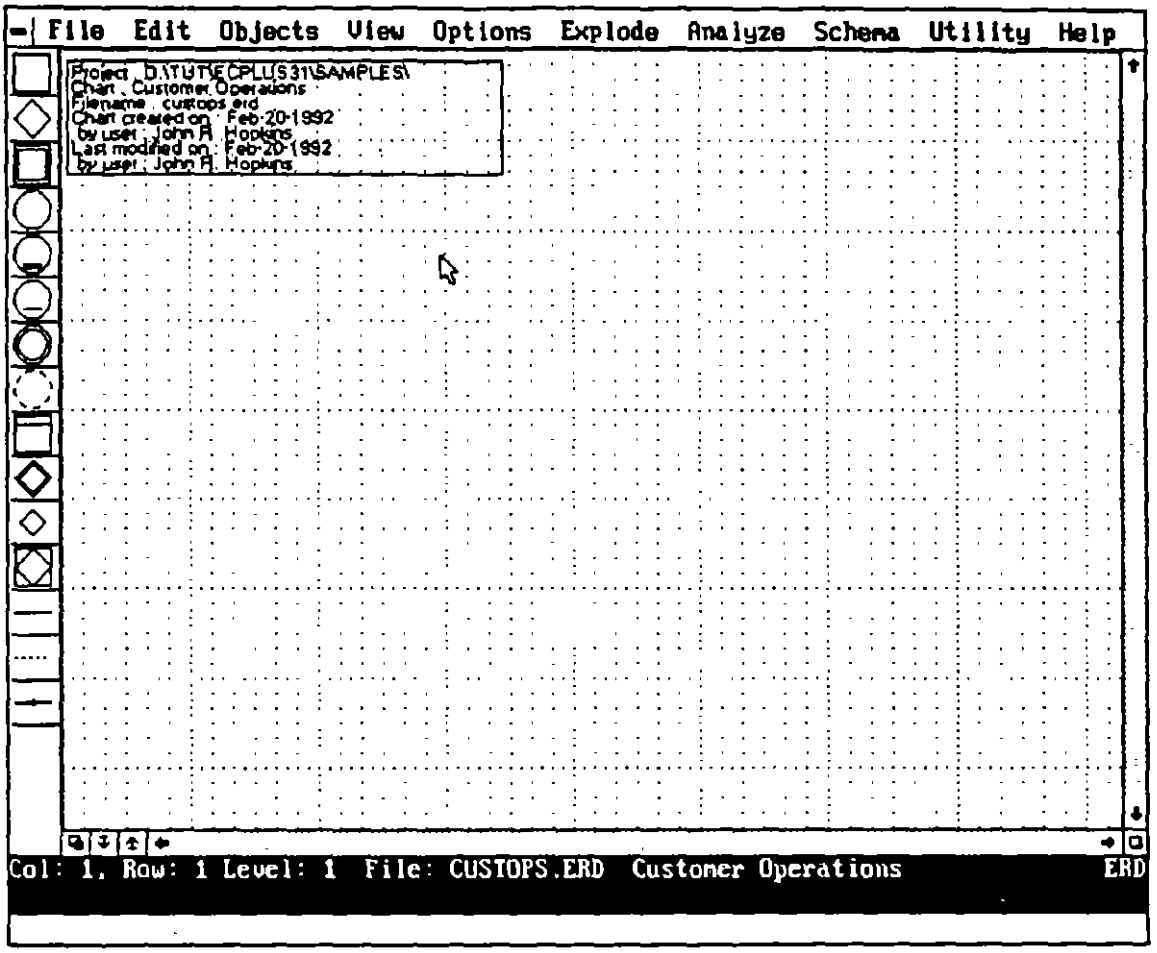


Figure 6.14 New ERD Chart

18. Select the 'Up a Level' option from the Explode menu, or press CTRL+U, or click on the explode up icon, to move back to the parent chart (TUTOR1.DFD).

19. You are asked to confirm if you want to save the new chart. Click on the 'No' option or press the ESC key.

You have just established a link between the Customer external entity on the original diagram to an ERD.

Establishing a Link to a Text File

A text file can also be specified as a child object. This allows you to explode a chart object to a note file, primitive process specification, or control specification in textual form. A text file is an ASCII file with extension .TXT, .DOC, .PPS, or .CTS. When you explode from a chart object to a text file, a text editor is also loaded.

This approach will also enable you to access an external documentation tool rather than a text editor, such as a flow charter, and pass the name of a file to it. This is described in more detail in Chapter 9 of the Chart Editor User's Guide.

Before you can explode down to a text file you must first select the default text editor as follows:

1. Choose the 'Set Default Editor' option from the Options menu.

2. If a text editor has already been chosen, its name will be displayed on the prompt line at the bottom of the screen, in the form of a full pathname and filename.

   A text entry box will appear on screen, as shown in Figure 6.21, and contains the specification (drive, pathname and filename) of the currently chosen text editor, if any. Initially it will be blank.

3. Assuming that you intend, for this example, to use the default text editor EDLIN.EXE, supplied with your version of MS-DOS, enter the text:

   C:\DOS\EDLIN.EXE

   into the text entry box and then click on the 'Yes' option button.

   Now, whenever you explode down to a text file, this (primitive) text editor will be used.

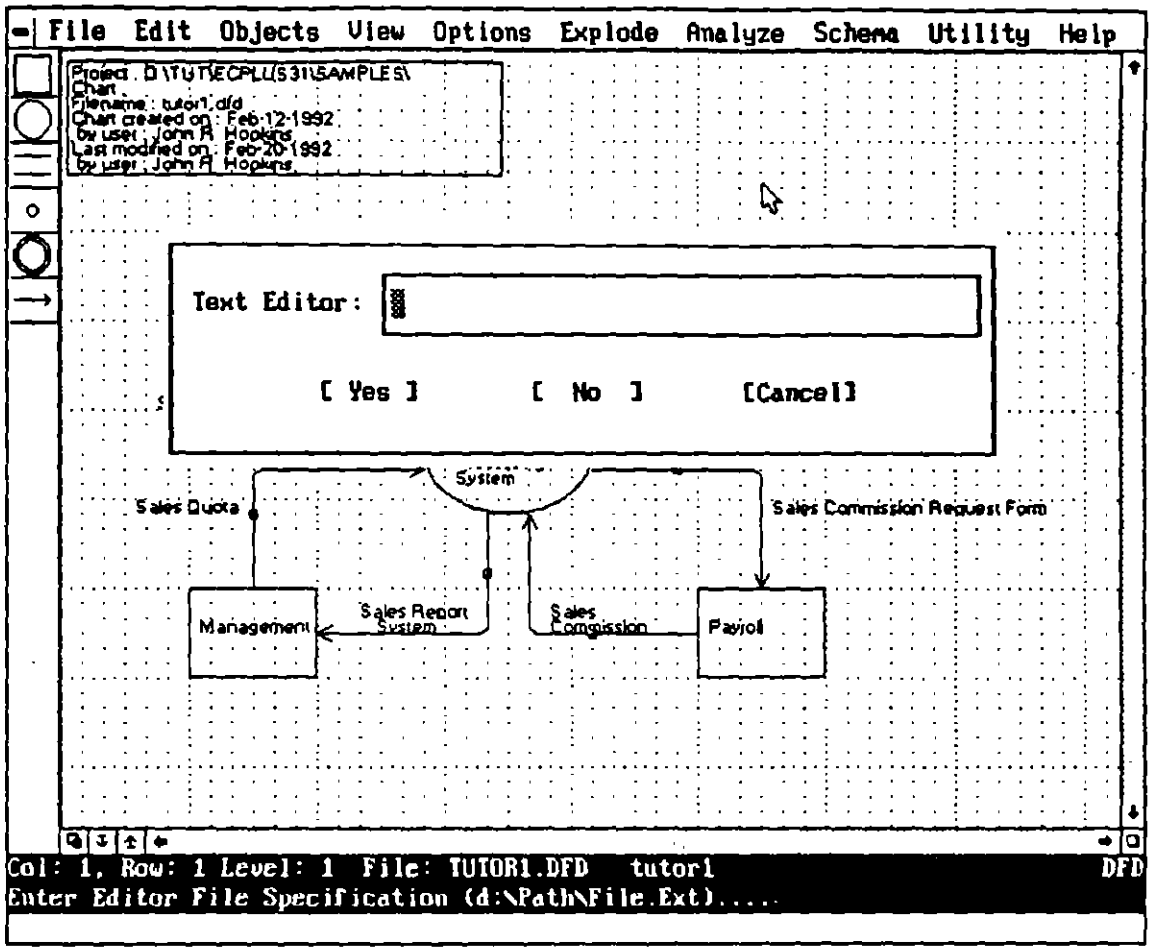


Figure 6.15 Enter Path to Text File Editor Dialog Box

Alternatively, you can specify your own particular favorite text editor such as Brief, the DOS 5.0 editor, etc., or even the name of your word processor. For further details on how to do this, refer to Chapter 9 of the Chart Editor's User's Guide.

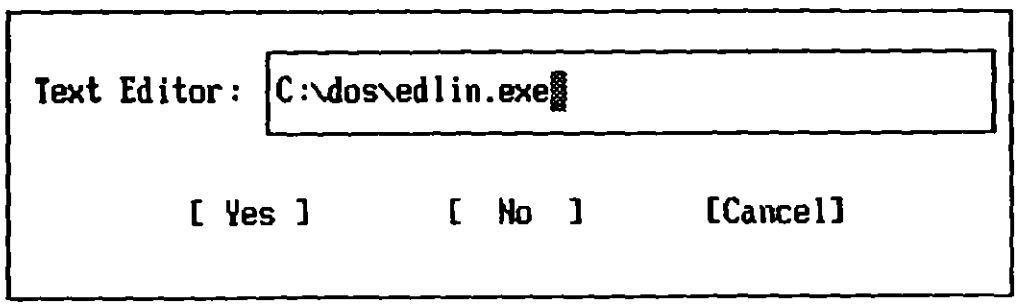


Figure 6.16 Enter the Path to the Text Editor Executable File

To link a chart object to a text file, use the Explodes To attribute fields in the data dictionary entry screen of the object just as you set up a link to a chart file previously.

Load the TUTOR1 chart, if it is not already loaded.

To set up a link to a text file:

1. Select the external entity symbol labeled "Management".

2. Choose the 'Identify Object' option from the Edit menu. You will be prompted for an ID for the object, which will be 'Management' by default. Click on the 'OK' option to use that identifier, as shown in Figure 6.17.

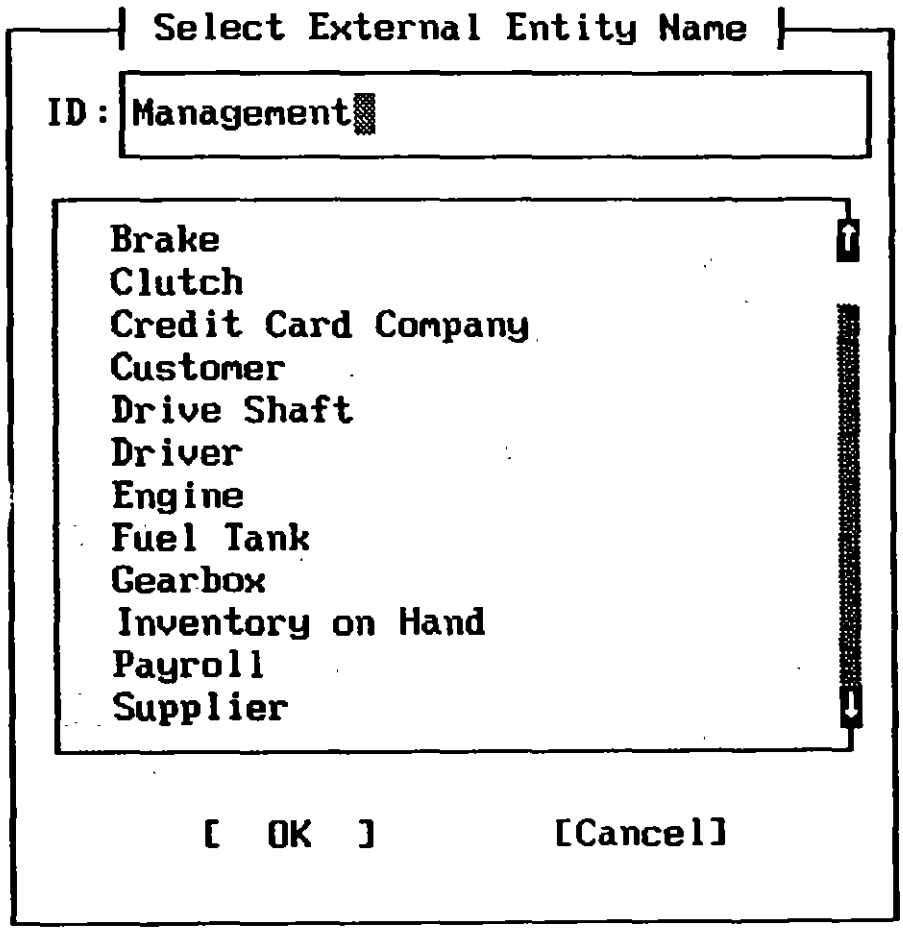


Figure 6.17 Entered Chart Object Name in the Text Entry Box

3. The DDE screen will appear. The Object Explodes To (Type, Name) fields will be blank.

4. Select the Object Explodes To - Type attribute field by clicking on the up/down arrow icon, as shown in Figure 6.18.

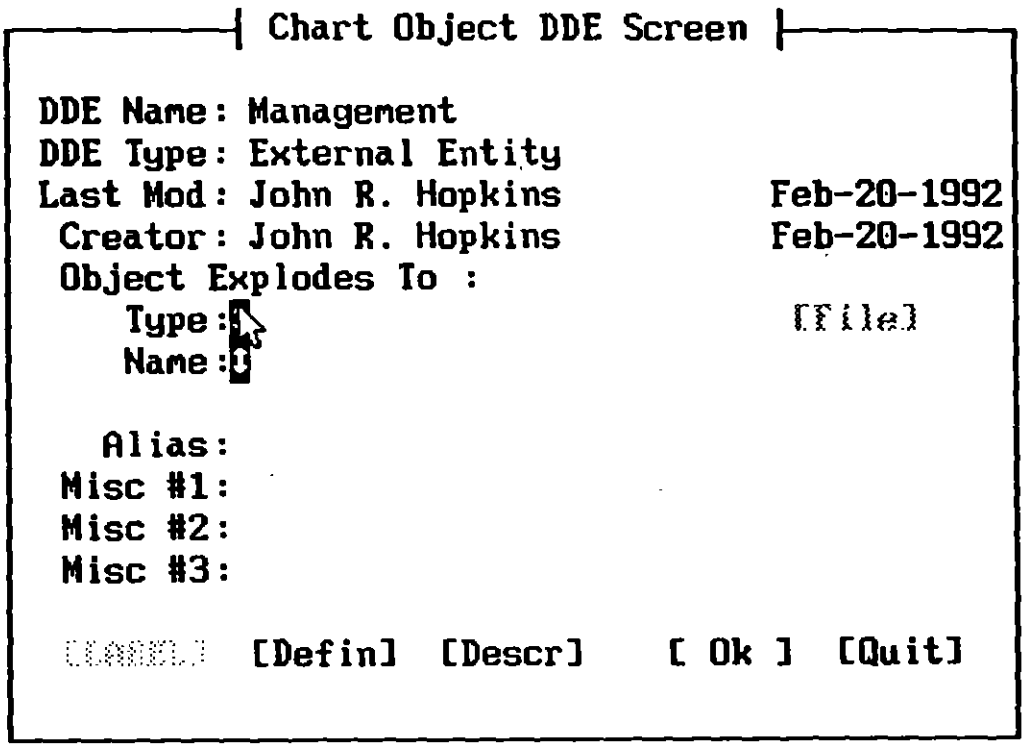


Figure 6.18 Cursor Poised over the Explodes To - Type Field

A dialog box will appear containing the list of valid child types, as shown in Figure 6.3.

5. Choose the 'txt' entry by clicking on it, then choose 'OK.' The text 'txt' will appear immediately to the right of the Explodes To - Type field name.

6. Select the Explodes To - Name attribute field by clicking on the up/down arrow icon next to this field.

7. A dialog box will appear containing a list of available text file entries as shown in Figure 6.19. Type in "Management" as the identifier for the text file, then choose 'OK.'

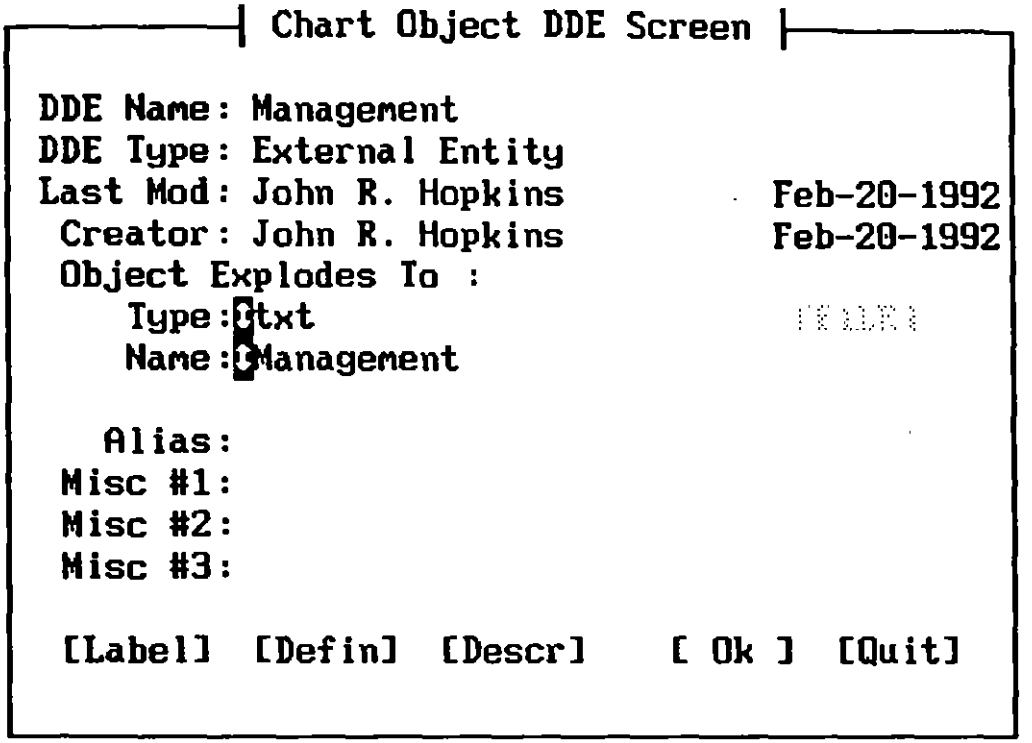


Figure 6.19 Selected Child Object Type and Name in DDE Screen

8. The text file name will appear immediately to the right of the Explodes To - Name field.

9. Select the [OK] button in the data dictionary screen to exit.

10. Press ENTER to place the label for the Management external entity.

You have just finished describing the link to the child chart object (a named text file) in the data dictionary not the text file name itself.

To provide a filename for the text file to be entered in the data dictionary:

1. The chart object should still be selected. If it is not, re-select it by clicking on it.

2. Select the 'Down a level' command from the Explode menu.

3. A dialog box will appear prompting you to save the chart. If so, choose the 'Yes' option.

4. A dialog box will appear prompting you to confirm creation of the data dictionary entry for the text file as shown in Figure 6.20. Choose the 'Yes' option.

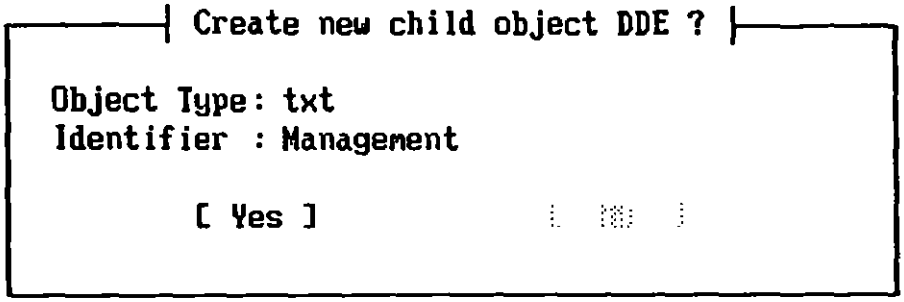


Figure 6.20 Prompt to Create DDE for Text File

5. A dialog box will appear prompting you to confirm creation of the new text file. Choose the 'Yes' option.

6. A dialog box will appear prompting you for the name of the text file. Enter "Mgmt.txt" as shown in Figure 6.21, then choose the 'Yes' option.

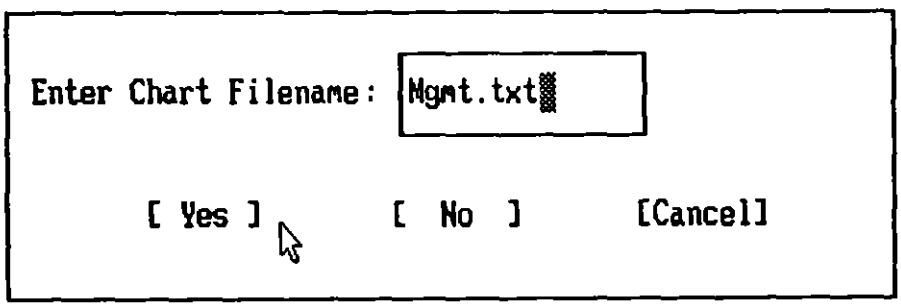


Figure 6.21 Enter Text Filename

7. The chosen editor will be run and you will be able to access the text file named MGMT.TXT.

8. To return to the chart editor, exit or quit from your text editor.

Establishing a Link to a Record

When a chart object is to be associated with a data structure definition, you link it to a Record definition that allows you to define a data structure.

When you establish a link to a record, you need to establish a data dictionary entry for the Record, just as you did for chart objects and text files. The components of a record are the specific data fields and are represented by elements.

In this section, you will load the TUTOR1.DFD chart and establish a link to a record used for information about Payroll. The components of this record define data such as the Employee Number, Name, Position, and so on.

To establish a link to a Record:

1. Load the TUTOR1.DFD chart.

2. Select the external entity symbol labeled "Payroll" by clicking on it.

3. Select the 'Down a Level' option from the Explode menu, or press CTRL+D, or click on the explode down icon.

4. A dialog box pops up listing available child object types as shown in Figure 6.9.

5. Select the 'Record' option by clicking on it.

6. Click on 'OK.'

7. A dialog box is displayed listing the identifiers for Record object types in the current project data dictionary.

8. Enter "Payroll" as the Record identifier in the text entry box at the top of the dialog box, as shown in Figure 6.22.

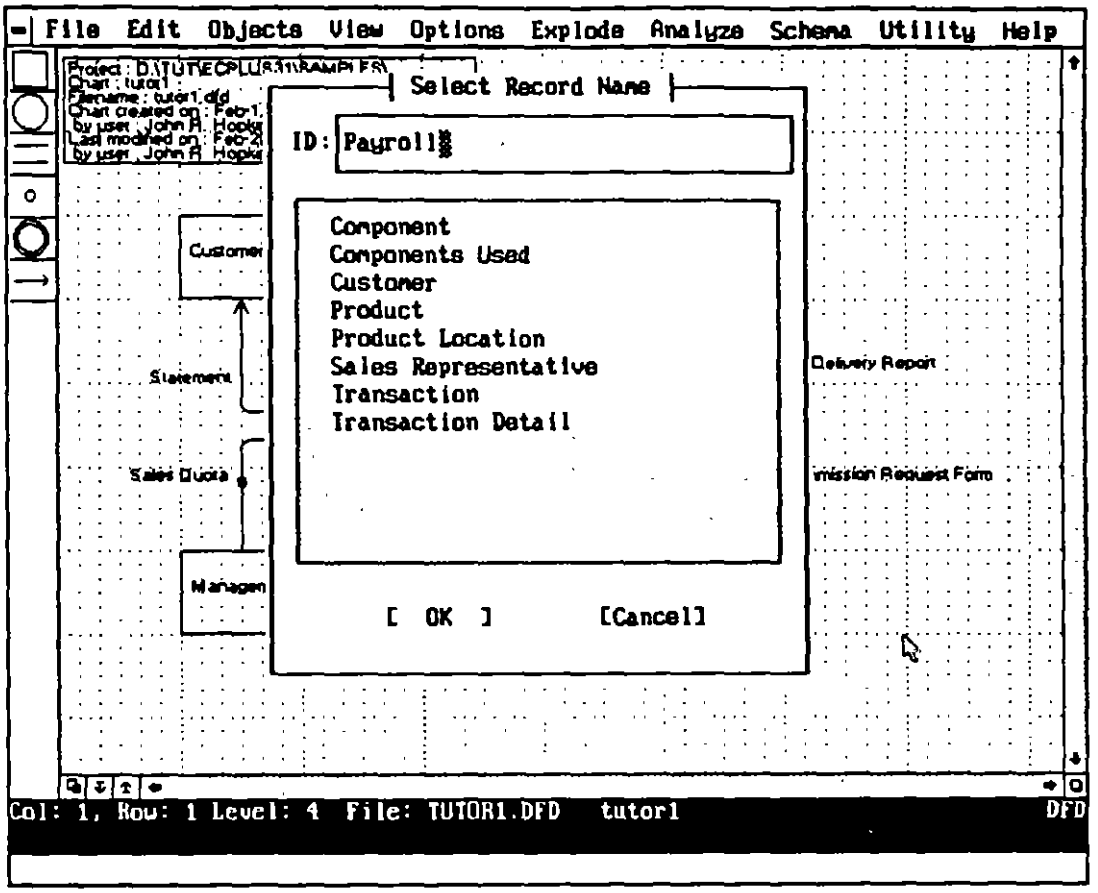


Figure 6.22 Entered Linked Record Name

9. Click on 'OK.'

So far you have established a link from the chart object to a named Record data dictionary entry. Next you will create the DDE and then access it.

10. A dialog box pops up prompting for confirmation to create the Payroll Record data dictionary entry, as shown in Figure 6.23.

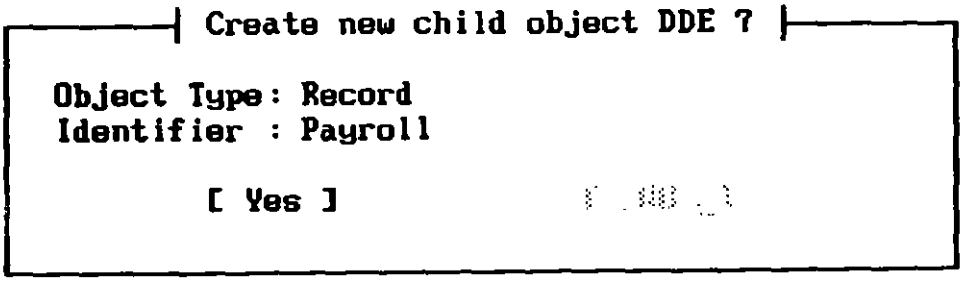


Figure 6.23 Prompt to Create Record DDE

11. Click on 'Yes.'

12. The Record definition data entry screen is then displayed, as shown in Figure 6.24.

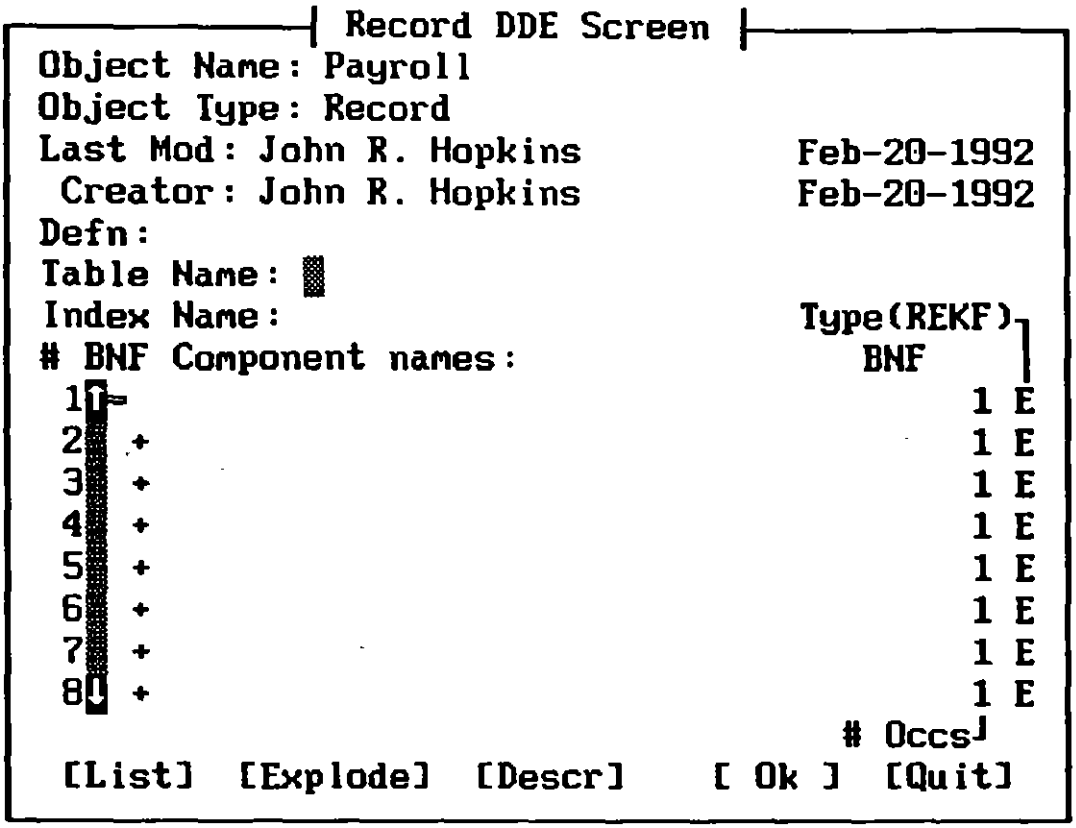


Figure 6.24 New Record Definition Screen

You have just defined a link between a chart object and a Record and have created a new record definition which is ready to be worked on. Take a few minutes to understand the fields associated with the Record definition data entry screen. The Object Name, Object Type, Last Mod, Creator, Defn, Table Name, and Index Name fields should be familiar to you from the chart object data dictionary entry screen.

The remaining fields are used to define the record components. A record can contain up to 128 such components, each of which is defined in detail by linking it in turn to an Element entry in the data dictionary.

Backus-Naur Operators

The Backus-Naur operator fields are the first and third fields (on either side of the component field). You can use these operators to specify how the record components are related. Allowable entries are; +(*{[(left side) and |]}*)+ (right side).

Component

This field defines a particular record component. The component name can be up to 32 characters in length. This is the name of an element defined in the data dictionary.

#Occs

The # Occs (Number of Occurrences) field determines how many times a particular component is to occur in the record. The default number of occurrences for all components is one.

Type

The Type field specifies the type of component. The available component types are indicated by "(REKF)". These letters identify the following component types:

- Record
- Element
- Key Field (an Element)
- Foreign Key Field (an Element)

The default type for all components is (E)lement.

The numbers down the left side of the Record Definition screen show that you can specify up to 128 components for a Record.

To enter the record components:

1. Click on the first position in the Component field, or press the ENTER key.

2. Type "Employee Number". This number will uniquely identify each Sales Representative.

3. Click on the Type field, or press the ENTER key three times.

4. Type "K" to replace the default entry of "E." The Employee Number component will be a key field.

5. Press ENTER to move to the next record component field (row #2).

6. Use Figure 6.25 to specify the remaining Record component definitions.

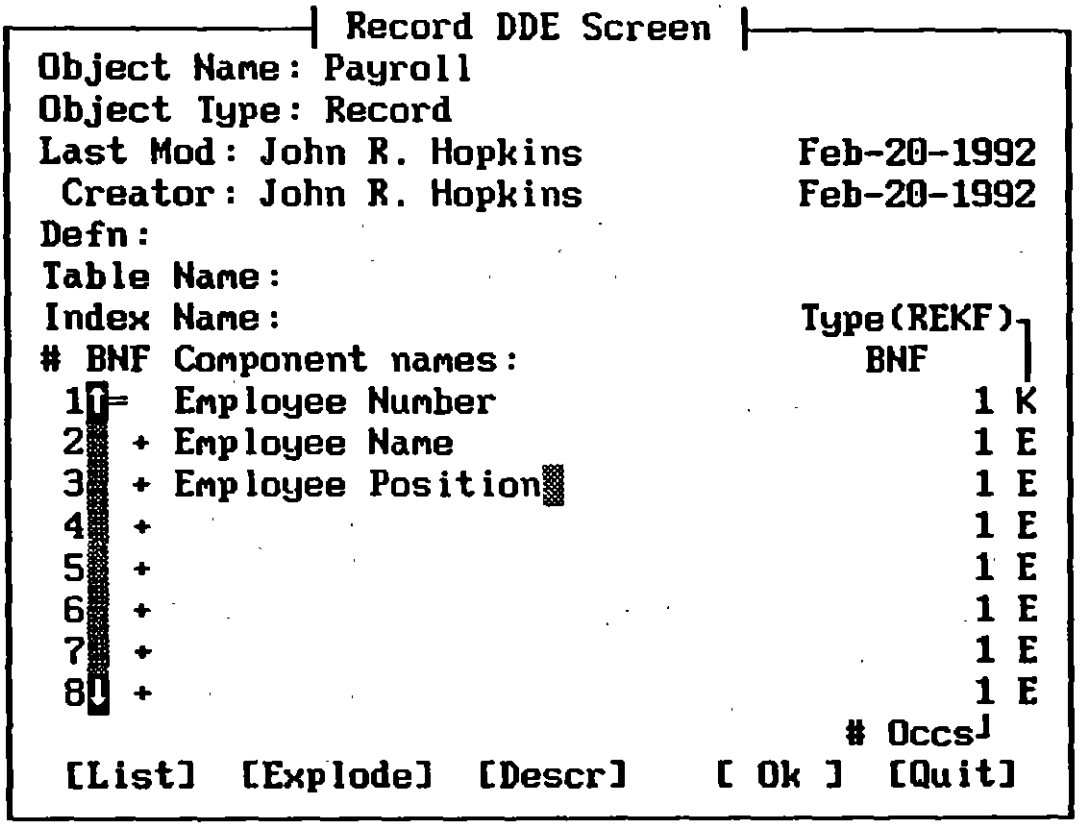


Figure 6.25 Completed Record Definition

At the bottom of the Record DDE screen are selection boxes to save or cancel the definition, as well as additional functions. The 'OK' and 'Quit' selections should already be familiar to you. The remaining functions are described below:

List

The List button accesses a list of any record components already defined in the data dictionary. Click on the [List] button to display the list as shown in Figure 6.26.

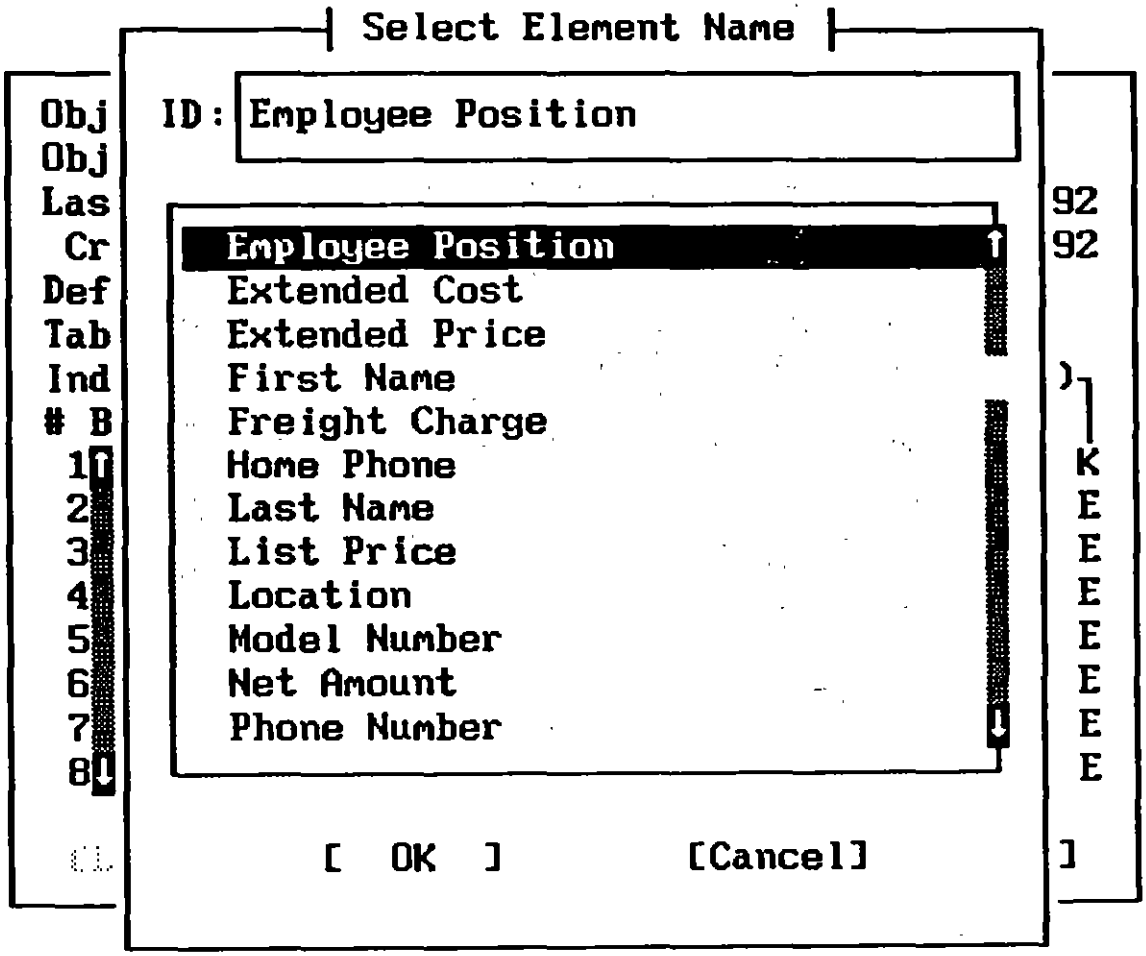


Figure 6.26 Display Element DDEs Using the [List] Option

Descr

The Descr button pops up a text entry box for you to enter a short description (up to 254 characters) of the purpose of the record.

1. Click on the [Descr] button to display the text entry box.

2. Enter "This record contains the employee number, name, and position." in the record description field as shown in Figure 6.27.

3. Press the F3 key to exit the description field.

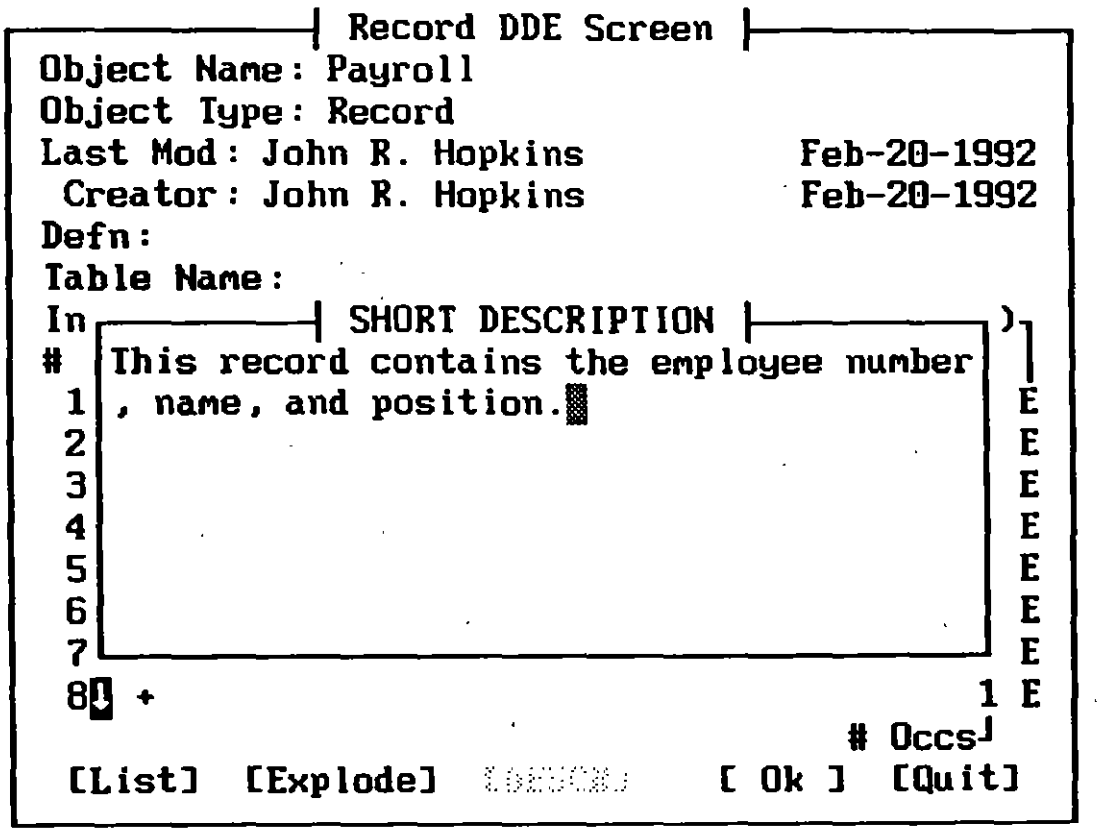


Figure 6.27 Text Entry Box for the Record Short Description

Explode

The Explode button enables you to create a data dictionary entry for a record component, known as an Element, and access the data dictionary entry screen for that Element.

To define the record components as Elements in the data dictionary:

1. Click on the 'Employee Number' component.

2. Click on the [Explode] button.

3. You may be prompted to verify that you want to create a data dictionary entry for the Element type component named Employee Number, as shown in Figure 6.28.

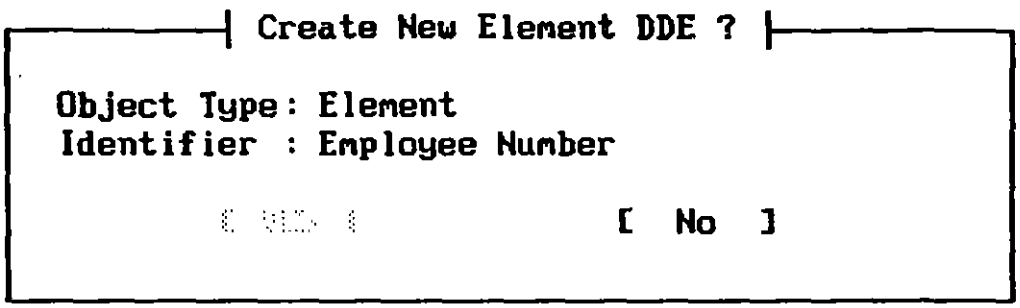


Figure 6.28 Prompt to Confirm DDE Creation for Chosen Element

4. If so, click on 'Yes.'

5. The data dictionary entry screen for the Element named "Employee Number" is displayed as shown in Figure 6.29.

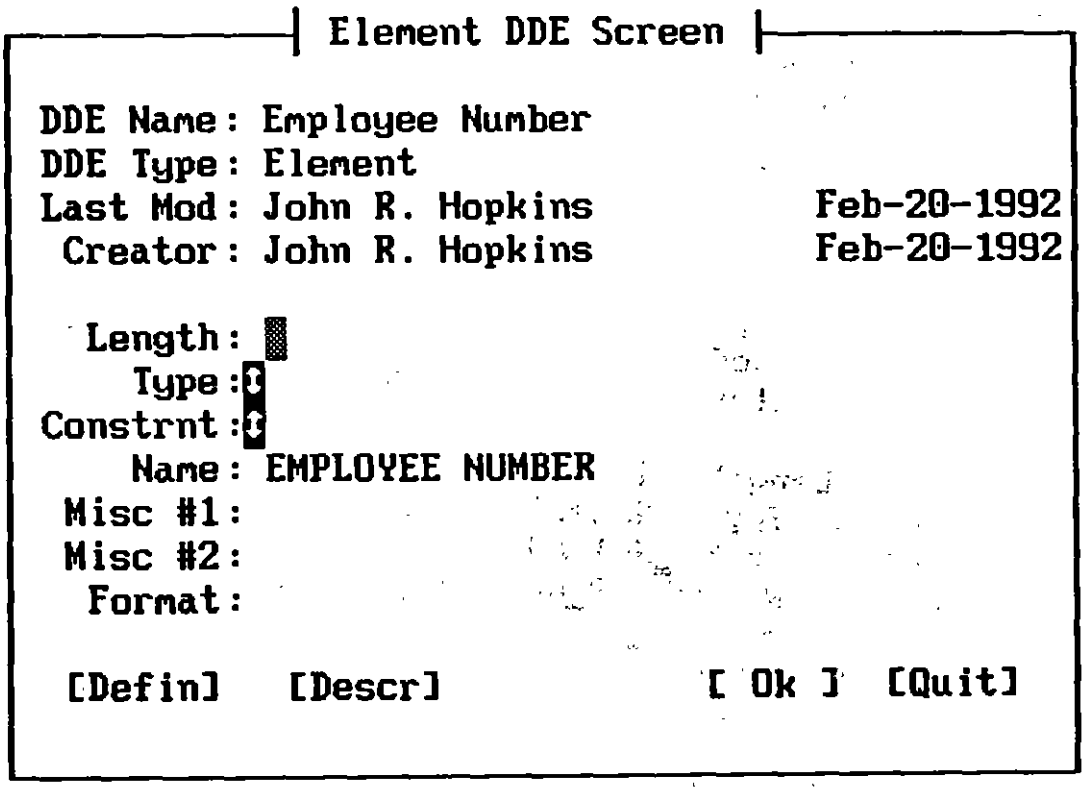


Figure 6.29 Data Dictionary Entry Screen for an Element

6. Click on the 'Length' field and enter "4".

   This means that the Employee Number can be up to 4 characters in length.

7. Click on the 'Type' field and type "Numeric".

   This means that the data type for the Employee Number is a number in the range of 0 to 9999.

8. Click on the 'Constraint' field and type "NOT NULL."

9. Click on the 'Format' field and type "9999".

   This is the format in which the data would be entered (digits only).

10. Click on the Definition field and type "A number uniquely identifying an employee". Then press the F3 key.

    This defines the purpose of the Employee Number element.

11. Click on the 'Name' field and type "EMP_NUM". This assigns a field name for the Employee Number field.

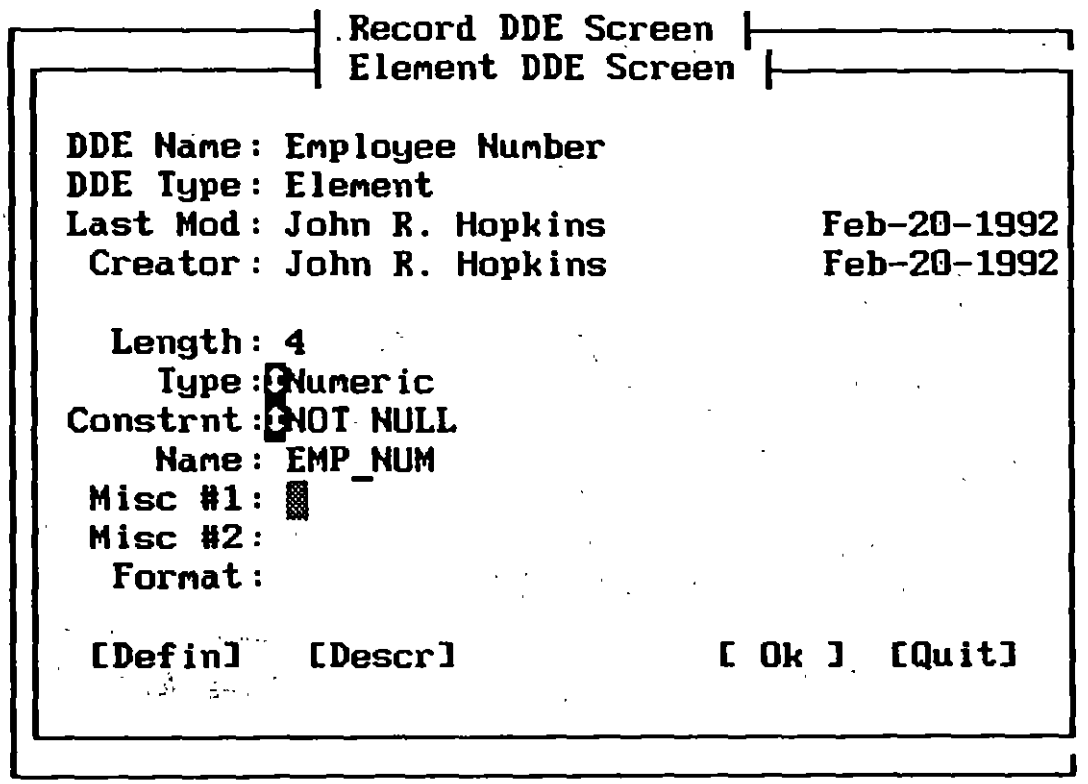


Figure 6.30 Complete Employee Number Element Definition

11. Click on 'OK.'

    The data dictionary entry for the Employee Number element is saved, and the Record definition screen is redisplayed.

12. Click on 'OK' to return to the chart editor.

When you have finished, you can then explode down to the completed Payroll record from the Payroll symbol.

To explode down to the Payroll record from the Payroll object:

1. The Payroll symbol should still be selected. If not, select it by clicking on it. Select the 'Down a Level' option from the Explode menu, or press CTRL + D, or click on the explode down icon.

2. You are prompted to save the current chart (if it has changed since the last save). If so, Click on 'Yes.'

3. The Payroll Record data entry screen is displayed. This will look the same as it did when you entered it as shown in Fig. 6.25.

4. Click on 'Quit' or press ESC to remove the record definition screen and return to the chart editor.

Alternatives

Explode menu

Top Level

Select the Top Level command to explode up to the highest level chart in a linked set of charts (or to the chart assigned as the top level chart, using the Define Top Level command).

For example, if you linked an object on the MODIFY chart to the ECPCONTX chart, you could select the Top Level command to explode up to the TUTOR1 chart.

Define Top Level

You can define any of the charts in a linked set of charts to be the top level chart. This will then be the chart you explode up to when you select the Top Level command.

For example, you could define the MODIFY chart to be the top level chart, rather than the TUTOR1 chart (the default, highest level chart).

Show Explosions

Select the Show Explosions command to indicate the object(s) on the TUTOR1 chart for which you have defined a link. This helps you to remember which objects are linked. The objects that have a defined link will be highlighted.

Update Explosions

The Update Explosions command updates the display of linked objects when you have the Show Explosions option turned on. This is useful when you define new linked objects and want to have them shown as such.

For More Information

See "Chart Linking and Decomposition" in the EasyCASE Plus Chart Editor User's Guide.

EPILOG

Congratulations! You have just finished the EasyCASE Plus Beginner's Tutorial. You have used the basic functions of the EasyCASE Plus chart editor program to create and modify charts, describe chart objects to the data dictionary, and link chart objects to child charts, records and text files. EasyCASE Plus has many more functions than could be covered in this beginner's tutorial. For more information about the Chart Editor, please refer to the Chart Editor User's Guide.

If you want to use the product to assist you in the design of relational database systems, using entity relationship diagrams (ERDs) and the data dictionary, it is recommended that you use the optional Schema Generator product to produce the resulting schemas. How to do this, in the form of a more advanced tutorial, is provided in the Schema Generator User's Guide and Tutorial.

The data dictionary stores the descriptions you gave to the charts, chart objects, records, etc. You can manipulate this data outside of the chart editor by using the supplied Data Dictionary and Reports Manager (DDRM) utility program. You can also generate specific chart and data dictionary reports using the DDRM. The DDRM can be accessed via the chart editor's Utility menu, or run the as a stand-alone application. For more information about the Data Dictionary and Reports Manager, please refer to the Data Dictionary and Reports Manager (DDRM) User's Guide.

You can use the optional Analysis Manager program to check and verify your charts and data dictionary. This utility program allows you to check your diagrams against a set of methodology, usage, and layout rules. It also allows for level balancing linked diagrams, checking record and element definitions, and even provides a "where used" analysis capability. For more information, please refer to the Analysis Manager User Reference.

To effectively use EasyCASE Plus, you should first be familiar with the software design methodologies supported by this CASE tool. EasyCASE Plus does not try to teach you about methodologies. However, it does supply the means whereby you can implement the most common, widely documented, well established methodologies. For more information about the methodologies, diagram types and symbologies supported by EasyCASE Plus, please refer to the provided Methodology Guide, which also contains a bibliography of popular software design methodology books.