UNIVERSIDAD POLITECNICA DE MADRID

FACULTAD DE INFORMATICA

TESIS DOCTORAL

DISEÑO DE UNA METODOLOGIA CASE DE DESARROLLO DE SOFTWARE

Autor: Antonio de Amescua Seco

Directores: Gonzalo Cuevas Agustín

Jose Luis Maté Hernández

Madrid, Febrero 1990

DEPARTAMENTO: LENGUAJES, SISTEMAS
INFORMATICOS E
INGENIERIA DE SOFTWARE

FACULTAD DE INFORMATICA

UNIVERSIDAD POLITECNICA DE MADRID

TESIS DOCTORAL

DISEÑO DE UNA METODOLOGIA CASE DE DESARROLLO DE SOFTWARE

Autor: **Antonio de Amescua Seco**
(Licenciado en Informática)

Directores: **Gonzalo Cuevas Agustín**
(Doctor en Informática)
Jose Luis Maté Hernández
(Catedrático de Universidad)

Madrid, Febrero 1990

TESIS DOCTORAL

DISEÑO DE UNA METODOLOGIA CASE DE DESARROLLO DE SOFTWARE

Autor: Antonio de Amescua Seco

Directores: Gonzalo Cuevas Agustín

Jose Luis Maté Hernández

Tribunal Calificador:

Presidente:	**D. Manuel Collado Machuca**
Vocales:	**D. Francisco Trigueros Ruiz**
	D. José Blat Gimeno
	D. Juan Pazos Sierra
Vocal Secretario:	**D. Jose Luis Morant Ramón**

Calificación:

Madrid, de de 1990

DEDICATORIA

A Mercedes y Carmen, porque sin ellas no hubiera podido ni comenzar, ni desarrollar, ni terminar esta Tesis.

AGRADECIMIENTOS

Deseo expresar mi profunda gratitud a todas aquellas personas que me han ayudado y facilitado la ardua tarea que supone la elaboración y conclusión de una tesis. Especialmente al Ilmo. Sr. D. Jose Luis Maté Hernández, actual Decano de la Facultad de Informática de la U.P.M. y a D. Gonzalo Cuevas Agustín, Profesor Titular de la Facultad de Informática de la U.P.M. quienes dirigieron este trabajo de investigación y con cuya ayuda conté desde el primer momento.

Asimismo quiero agradecer la ayuda incondicional de mis amigos y compañeros de Facultad Dª. Genoveva López Gómez, D. Luis García Sánchez, y D. Luis Fernández Sanz.

Mención aparte y especial merece mi mujer que con su ayuda y apoyo moral ha hecho posible que pudiera realizar este trabajo de investigación.

Por último, no quiero dejar de mencionar a la Facultad de Informática de Madrid, que me ha proporcionado los recursos y medios materiales para la realización de esta Tesis.

RESUMEN

En la actualidad existe una gran expectación ante la introducción de nuevas herramientas y métodos para el desarrollo de productos software, que permitirán en un futuro próximo un planteamiento de ingeniería del proceso de producción software. Las nuevas metodologías que empiezan a esbozarse suponen un enfoque integral del problema abarcando todas las fases del esquema productivo. Sin embargo el grado de automatización conseguido en el proceso de construcción de sistemas es muy bajo y éste está centrado en las últimas fases del ciclo de vida del software, consiguiendose así una reducción poco significativa de sus costes y, lo que es aún más importante, sin garantizar la calidad de los productos software obtenidos.

Esta tesis define una metodología de desarrollo software estructurada que se puede automatizar, es decir una metodología CASE. La metodología que se presenta se ajusta al modelo de ciclo de desarrollo CASE, que consta de las fases de análisis, diseño y pruebas; siendo su ámbito de aplicación los sistemas de información.

Se establecen inicialmente los principios básicos sobre los que la metodología CASE se asienta. Posteriormente, y puesto que la metodología se inicia con la fijación de los objetivos de la empresa que demanda un sistema informático, se emplean técnicas que sirvan de recogida y validación de la información, que proporcionan a la vez un lenguaje de comunicación fácil entre usuarios finales e informáticos. Además, estas mismas técnicas detallarán de una manera completa, consistente y sin ambigüedad todos los requisitos del sistema.

Asimismo, se presentan un conjunto de técnicas y algoritmos para conseguir que desde la especificación de requisitos del sistema se logre una automatización tanto del diseño lógico del Modelo de Procesos como del Modelo de Datos, validados ambos conforme a la especificación de requisitos previa.

Por último se definen unos procedimientos formales que indican el conjunto de actividades a realizar en el proceso de construcción y cómo llevarlas a cabo, consiguiendo de esta manera una integridad en las distintas etapas del proceso de desarrollo.

ABSTRACT

Nowdays there is a great expectation with regard to the introduction of new tools and methods for the software products development that, in the very near future will allow, an engineering approach in the software development process. New methodologies, just emerging, imply an integral approach to the problem, including all the productive scheme stages. However, the automatization degree obtained in the systems construction process is very low and focused on the last phases of the software life-cycle, which means that the costs reduction obtained is irrelevant and, which is more important, the quality of the software products is not guaranteed.

This thesis defines an structured software development methodology that can be automated, that is a CASE methodology. Such a methodology is adapted to the CASE development cycle-model, which consists in analysis, design and testing phases, being the information systems its field of application.

Firstly, we present the basic principles on which CASE methodology is based. Secondly, since the methodology starts from fixing the objectives of the company demanding the automatization system, we use some techniques that are useful for gathering and validating the information, being at the same time an easy communication language between end-users and developers. Indeed, these same techniques will detail completely, consistently and non ambiguously all the system requirements.

Likewise, a set of techniques and algorithms are shown in order to obtain, from the system requirements specification, an automatization of the Process Model logical design, and of the Data Model logical design. Those two models are validated according to the previous requirement specification.

Finally, we define several formal procedures that suggest which set of activities to be accomplished in the construction process, and how to carry them out, getting in this way integrity and completness for the different stages of the development process.

INDICE

INDICE

1.- INTRODUCCION

1.1.- JUSTIFICACION DE LA INVESTIGACION

Los productos informáticos vienen adoleciendo de:

- retrasos considerables en la planificación
- poca productividad
- elevadas cargas de mantenimiento
- demandas cada vez más desfasadas con las ofertas
- baja calidad y fiabilidad del producto
- dependencia de los realizadores

Ello ha venido originado principalmente por una falta de:

- formalismo y metodología
- herramientas de soporte y
- administración eficaz

En el momento actual está surgiendo una gran expectativa ante la evolución de la ingeniería del software, al ir apareciendo nuevos métodos y herramientas formales que van a permitir en el próximo futuro un planteamiento de ingeniería del proceso de producción de software. Dicho planteamiento vendrá a solucionar, o paliar al menos, la demanda creciente por parte de los usuarios de software, permitiendo dar respuesta a los problemas

antes planteados de:

- administración
- calidad
- productividad y
- fácil mantenimiento

Este último, es uno de los principales problemas de la mayoría de las instalaciones, llegando a suponer un coste mayor al 60% del coste del ciclo de vida del producto software [KUN89].

Las nuevas metodologías que empiezan a esbozarse suponen un enfoque integral del problema, es decir abarcan todas las fases del esquema productivo, así como los aspectos de administración (planificación y control, medida y seguimiento, administración de configuración y garantía de calidad) que en su mayoría no se tenían en cuenta en los desarrollos tradicionales. Todo ello supone mejoras importantes y necesarias en todos los dominios. En particular son fundamentales la reducción de costes y plazos, así como la calidad del producto final.

La tendencia predominante de hoy día es alcanzar un alto nivel de automatización en el proceso de construcción de sistemas para de este modo reducir sus costes. Pero

también existe una razón más, fundamental para la automatización, y es que el cerebro humano es limitado, lo cual hace que se produzcan muchos errores cuando se deben manejar gran cantidad de detalles complejos. Asimismo, en muchos casos se hace difícil manejar técnicas muy precisas o de tipo matemático. Sin embargo si automatizamos estas técnicas podremos llegar a aplicar métodos cada vez más rigurosos consiguiendo así la calidad requerida.

Ultimamente se está produciendo un cambio muy importante en las herramientas software con la aparición de aquellas que automatizan distintas tareas del proceso de desarrollo de software, y que además ofrecen la posibilidad de ser utilizadas en ordenadores personales, con lo que reducen muy significativamente el proceso de producción de software, ya que el coste de un MIPS en una estación de trabajo es mil veces inferior al coste de un MIPS en un ordenador principal. Muchas de ellas están enfocadas hacia las fases de análisis y diseño, pudiéndose crear sistemas interactivamente.

Esta nueva filosofía de la automatización del software recibe el nombre de Ingeniería de Software Asistida por Ordenador. La comúnmente referida como tecnología CASE (Computer-Aided Software Engineering), podría ser definida

de una forma sencilla y simple como: "La automatización del desarrollo de software".

CASE propone una nueva aproximación al concepto de ciclo de vida del software, basada en la automatización. La idea básica que subyace en CASE es proveer un conjunto de herramientas bien integradas que automaticen todas las fases del ciclo de vida del software.

¿ Qué novedades aporta CASE frente a las tecnologías software tradicionales ?

Las tecnologías software tradicionales son de dos tipos: herramientas y metodologías. Las primeras, tal y como vemos en la figura 1.1, están dirigidas hacia las últimas fases del ciclo de desarrollo del software y no están integradas unas con otras, mientras que las segundas están enfocadas a las fases de análisis, diseño e implantación, y en este caso, o bien no están integradas o bien las reglas de transición de una fase a otra son ambiguas e incompletas.

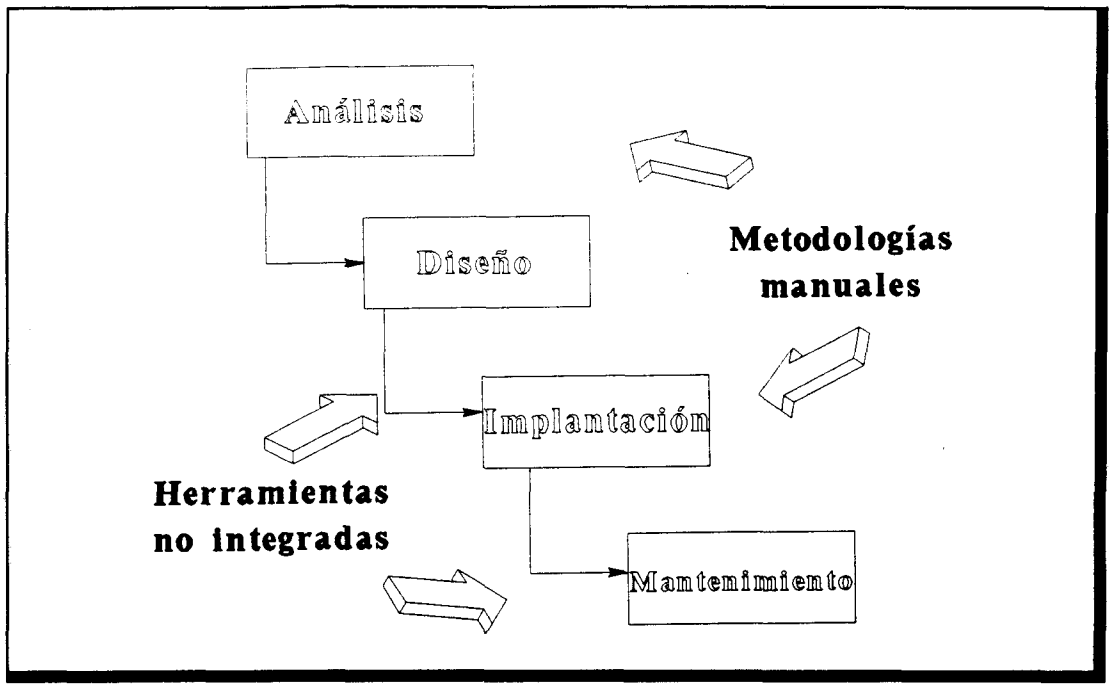


Figura 1.1 Tecnologías software tradicionales

La categoría de herramientas incluye una variedad de herramientas de tercera, cuarta, y más recientemente, de quinta generación.

La categoría de metodología software incluye metodologías de desarrollo del software manuales tales como análisis estructurado, diseño estructurado y programación estructurada. Dichas metodologías definen un proceso disciplinado paso por paso para desarrollar software. Pero al ser manuales, son demasiado tediosas y de una labor intensa, por lo que en la práctica casi nunca se siguen al nivel más detallado.

Por otra parte, es curioso observar cómo se cumple el refrán "En casa del herrero cuchillo de palo" en el caso de los profesionales que desarrollan software, pues a pesar de haber hecho bastante por automatizar el trabajo de los demás, se sientan en su mesa de trabajo y elaboran especificaciones y crean diseños de forma manual, con errores, inconsistencias y omisiones que podrían ser detectadas por un ordenador.

El hecho de facilitar la construcción y la consistencia de las aplicaciones y el asegurar el control sobre ellas de forma automatizada, para su posterior mantenimiento, será tan importante como las propias aplicaciones.

La tecnología CASE es una combinación de herramientas y metodologías software constituyendo un todo integrado. Además, CASE se diferencia de las tecnologías software iniciales en que está enfocada en su totalidad hacia el problema de la productividad y calidad software, y no sólo hacia soluciones de implantación.

1.2.- DESARROLLO DE LA INVESTIGACION

1.2.1.- DEFINICION DE OBJETIVOS

El objetivo de este trabajo de investigación es el siguiente: **"Definir una metodología integrada CASE que cubra todas las fases del ciclo de desarrollo del software. El ámbito de aplicación de la metodología a diseñar queda restringido a la automatización de sistemas de información."**

No se incluye en esta metodología el aspecto de *ingeniería hacia atrás* que permite llevar a cabo el mantenimiento de las aplicaciones tradicionales con métodos modernos.

Por *Sistemas de Información* debemos entender una red de informaciones formalizadas y estructuradas en función de las necesidades y posibilidades de una organización, cuya misión es suministrar a todos los elementos de dicha organización las informaciones necesarias para la adecuada realización de sus actividades.

Por metodología CASE se entiende *"una metodología estructurada que se puede automatizar y que define una aproximación disciplinada y de ingeniería para todos o*

algunos aspectos del desarrollo y mantenimiento software" [CLU89].

Las técnicas estructuradas, junto con el uso de herramientas soportadas por ordenador, son la base para el CASE, si bien hay que efectuar una serie de cambios en estas técnicas para obtener de este modo métodos rigurosos y formales que eviten la tediosa tarea de trabajar a mano.

Se podría pensar que las técnicas estructuradas no son necesarias en el mundo de los generadores de aplicaciones, o que el desarrollo de aplicaciones sin programadores implica no usar técnicas estructuradas, pero ocurre todo lo contrario. El problema básico del desarrollo de grandes sistemas software es el manejo de la complejidad, y de esto es de lo que en realidad se ocupan las técnicas estructuradas: nos permiten descomponer problemas complejos en pequeños problemas, convirtiéndolos así en problemas intelectualmente manejables.

Descomponiendo un problema en subproblemas (dominio de la complejidad mediante el análisis y programación estructurada) obtenemos el nivel óptimo de productividad, al mismo tiempo que mejoramos la calidad y la facilidad de mantenimiento del problema.

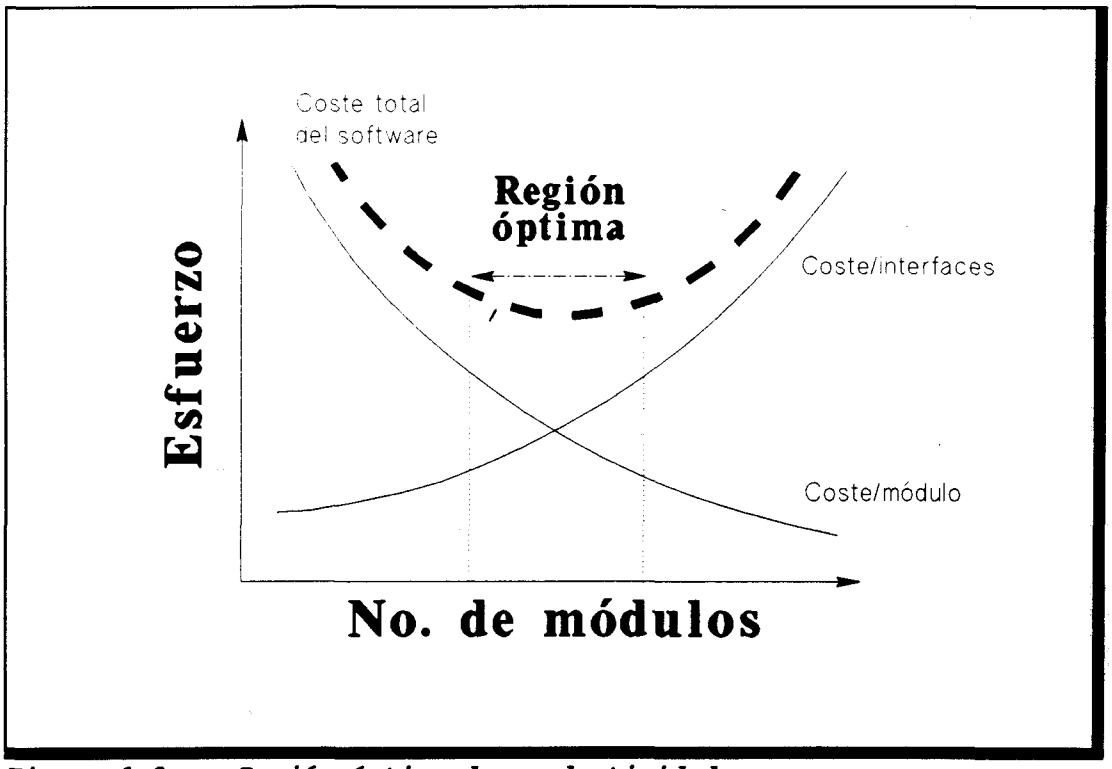


Figura 1.2 Región óptima de productividad

En el gráfico se muestra cómo el tamaño de los módulos no puede descomponerse indefinidamente debido al coste que genera los interfaces [PRE88] lo que da un intervalo aconsejable.

Con la metodología de desarrollo de software no solo lograremos mejorar la productividad del que lo desarrolla, mediante el empleo de técnicas estructuradas soportadas en herramientas CASE, sino que también conseguiremos un producto software de alta calidad, ya que se lleva a cabo

un proceso de producción sobre bases de ingeniería.

1.2.2.- DESARROLLO DEL PROYECTO

El logro de tal objetivo inicial implica la necesidad de realizar el siguiente conjunto de actividades:

1. Analizar y comparar las metodologías estructuradas existentes más empleadas y que estén ampliamente soportadas por herramientas CASE.

2. Determinar un conjunto de principios básicos en los que debe apoyarse la metodología CASE a diseñar.

3. Determinar el conjunto de técnicas a emplear por la metodología a desarrollar.

4. Definir la metodología CASE propuesta a nivel de fases, tareas y actividades de la misma así como de las técnicas que emplea para llevar a cabo tales actividades.

El resultado de cada una de las actividades anteriores y de sus objetivos y procesos intermedios se expone en el

presente documento, con la estructura y contenido que se describen en la siguiente sección.

1.2.3.- ESTRUCTURA DEL DOCUMENTO

Se presenta en primer lugar el "Estado de la Cuestión en Metodologías de Desarrollo de Software", desde el punto de vista de Análisis y Diseño, que es justamente el alcance que queremos dar a la metodología que vamos a desarrollar en esta investigación, dado que la propia filosofía CASE elimina la fase de implantación como se verá en la exposición. Se realizará asimismo una presentación de las metodologías que se van a analizar. El análisis consistirá en la exposición de cómo cada metodología define:

- las fases del ciclo de desarrollo
- los objetivos a lograr en cada fase
- las actividades a realizar en cada fase para lograr el objetivo marcado
- las técnicas que aplican para la realización de las actividades y una exposición de cada técnica empleada

A continuación se expone el desarrollo de la

"Metodología Integrada de Desarrollo Software basada en CASE", que en lo sucesivo denominaremos MIDES. Esta exposición está subdividida en las siguientes partes.

En la primera parte se expone la base sobre la cual se apoya la metodología MIDES. Esta base está constituida por una serie de principios que constituyen el pilar de la metodología MIDES. Estos principios son derivados del estudio de:

- el concepto de modularidad
- las técnicas estructuradas
- el uso de bases de datos
- la ingeniería de software
- el diseño asistido por ordenador

Seguidamente se exponen todas las técnicas que se utilizarán en la metodología MIDES. Las técnicas han sido seleccionadas de acuerdo a:

- la posibilidad de ser automatizadas
- la adecuación a los principios de la metodología establecidos

A continuación se describen las fases que componen MIDES. Para cada fase se establecen los objetivos a lograr y se detallan los procedimientos concretos que hay que realizar, así como la técnica y formalismo a emplear.

Por último se realiza un apartado de futuras líneas de investigación.

ESTADO DE LA CUESTION

CAPITULO 2

2.- ESTADO DE LA CUESTION DE LAS METODOLOGIAS DE DESARROLLO DE SOFTWARE

2.1.- EVOLUCION

Las técnicas estructuradas abarcan desde métodos de programación (programación estructurada), hasta técnicas que incluyen el análisis y diseño de aplicaciones, como también métodos de prueba, conceptos de administración de proyectos y técnicas de documentación. El nacimiento de las técnicas estructuradas es el primer escalón para pasar de la construcción de programas de forma artesanal, a una forma que siga unos métodos de ingeniería, generando finalmente las bases para un desarrollo automatizado. Estas técnicas tienen unos conceptos generalmente aceptados y que se plasman en diferentes métodos más o menos formalizados y por tanto más o menos fáciles de automatizar.

Las técnicas estructuradas se introdujeron en la comunidad académica a finales de los años 60, pasando a ser dominio de la industria a fines de los 70, incorporadas a todo el conjunto de actividades que abarca el ciclo de vida de los sistemas. Estas técnicas iban dirigidas tanto a los aspectos de desarrollo como a los de administración, abarcando desde la solución de problemas de procedimientos

administrativos a la construcción de un programa en un lenguaje determinado.

La figura 2.1, muestra la secuencia de evolución que han seguido las técnicas estructuradas más conocidas.

2.1.1.- PROGRAMACION ESTRUCTURADA

En un comienzo la técnica estructurada se centró en la programación, las inquietudes se dirigieron a la forma del programa, cómo debía verse un programa, la relación entre su forma estática y su estructura dinámica de ejecución, la manera para llegar a que un programa fuera más comprensible, cómo controlar la complejidad en programas de gran tamaño.

La solución fue el establecimiento de normas referidas a la aplicación de las estructuras de datos y de control.

Comienzo de los 60

PROGRAMACION ESTRUCTURADA

* Convenciones de codificación estructurada
* Programación Top-Down
* Niveles de abstracción
* Refinamiento paso a paso

Mediados de los 70

DISEÑO ESTRUCTURADO

* Diseño estructurado de Yourdon-Constantine
* Metodología de diseño Jackson
* Metodología de diseño Warnier

Finales de los 70

ANALISIS ESTRUCTURADO

* Análisis estructurado DeMarco
* Análisi estructurado de Gane y Sarson
* Lenguaje de diseño de requerimientos

Comienzo de los 80

TECNICAS AUTOMATICAS

* Verificación axiomática de HOS
* Modelo de datos automático
* Lenguajes no procedurales
* Diagramas de acción

Finales de los 80

TECNICAS DE DISEÑO ASISTIDO POR ORDENADOR

* Estaciones gráficas de trabajo para diseño
* Editores de diagramas para lenguajes de cuarta generación
* Sistemas basados en reglas
* Especificaciones desde las cuales el código se genera automáticamente

Figura 2.1 Evolución de las técnicas estructuradas

2.1.2.- DISEÑO ESTRUCTURADO

A mediados de los años 70, la técnica estructurada se extendió a la fase de diseño. La idea de normalizar las cosas se aplicó al proceso de solución del problema como una forma de introducir organización y disciplina en el diseño de los programas.

De la misma forma en que las primeras técnicas estructuradas se concentraron en el nivel de instrucciones de programas, el diseño estructurado se enfocó sobre la base de una programación a alto nivel, usando el módulo de programa como el componente básico de construcción. Se estudió la relación entre la estructura real del problema y la de su programa, esto es: la forma debe seguir a la función.

Se refinó el concepto de modularidad, normalizando la estructura de un módulo de programa, restringiendo las relaciones entre módulos y estableciendo medidas en la calidad de los programas.

2.1.3.- ANALISIS ESTRUCTURADO

Cuando se descubrió que muchos de los problemas que presentaban los programas eran producto de definiciones de requerimientos deficientes, la tensión se centró en la fase de análisis. A finales de los años 70 se desarrollaron las técnicas estructuradas para análisis de sistemas y especificación de requisitos. Debido a esto se extendió rápidamente el uso de diagramas de flujos, y junto con ello se desarrollaron las técnicas de bases de datos.

La introducción de gran cantidad de técnicas estructuradas en el proceso de desarrollo de sistemas, hizo que éstas derivaran en metodologías, las cuales ofrecían estrategias y herramientas para la aproximación sistemática al desarrollo y construcción de sistemas. La diferencia entre estas metodologías radicaba en las funciones a ejecutar dentro del ciclo de vida y su forma de aplicación, pero todas compartían una filosofía común.

2.1.4.- TECNICAS AUTOMATICAS

A inicios de los años 80 hizo crisis la baja productividad de la programación. Se produce la explosión del mercado de los ordenadores, particularmente de los ordenadores personales y como consecuencia usuarios finales comenzaron a adquirir conocimientos de informática y a pedir el desarrollo de más aplicaciones. Los departamentos de proceso de datos, usando las metodologías antes indicadas, simplemente no podían desarrollar aplicaciones en forma rápida y se encontraban cada vez con más problemas de mantenimiento. El esfuerzo por aumentar la productividad llevó a la creación de nuevos lenguajes, generadores de informes, generadores de aplicaciones, herramientas de bases de datos, programas para ayudar a la toma de decisiones, herramientas para usuarios finales y una variedad de formas para crear especificaciones que permitieran generar código a partir de ellas en forma automática.

En esta etapa de evolución, el objetivo es lograr un alto nivel de automatización en el desarrollo de los sistemas, y para ello:

- se hace uso intensivo del ordenador como estación de trabajo para el diseño, aprovechando al máximo las ventajas que ofrece para esta actividad.

- se aprovecha la potencia y variedad de uso que ofrecen las técnicas estructuradas.

Los ordenadores se emplean para crear, editar, ampliar y modificar diagramas estructurados, y al mismo tiempo para mantener diccionarios, directorios y todo tipo de información necesaria para asistir al análisis y diseño. Pero el uso que se espera de los ordenadores es mucho mayor. Los ordenadores deben ayudar a la automatización del modelo de datos y procesos, y extraer subconjuntos de los modelos de datos y de procesos para el equipo de desarrollo que está dedicado a una parte del problema general.

También se espera que el ordenador se emplee para hacer verificaciones de los diseños que ya han sido creados, para esto se han de usar técnicas de verificación basadas en reglas de tipo matemático.

A partir del diseño ya verificado sí se pueden usar programas generadores de código, con los cuales se logra rapidez y se evitan los errores que se producen en el

proceso de programación convencional.

Las técnicas estructuradas que ahora son consideradas como tradicionales (Constantine, Yourdon, DeMarco, Jackson, Warnier) han proporcionado un avance importante al proceso del análisis, diseño y programación, pero no han llegado a la verificación de código sin errores. Por medio de la adaptación y refinamiento de las técnicas usadas por ellos podemos conseguir un alto nivel de automatización, con resultados de alta calidad, mejor administración de la complejidad, y un desarrollo de sistemas más rápidos.

2.1.5.- ANALISIS ORIENTADO A OBJETOS

A finales de los 80 surge una forma diferente de plantear y solucionar los problemas basada en el desarrollo en base a objetos. Una de las diferencias fundamentales es que, así como el análisis estructurado transforma la realidad contemplándola como un árbol de objetos interrelacionados, cuando ésta corresponde normalmente a una red, el diseño orientado a objetos permite representarla tal como es, normalmente en forma de red de objetos que se intercambian mensajes, manteniéndose cada objeto en su mayor parte oculto a los demás.

Otro aspecto fundamental de esta nueva técnica de análisis es el concepto de herencia, permitiendo definir metaobjetos, objetos, etc. (padres, hijos que heredan la información de sus antecesores). Esto último hace que, de momento, las técnicas de análisis y diseño orientados a objetos sean sensibles a las posibilidades de los lenguajes.

Uno de los aspectos más interesantes de destacar de esta técnica, aparte de las diferencias marcadas con respecto al análisis estructurado, es que soporta las bases para una efectiva producción de software reusable.

2.2.- METODOLOGIAS DE DESARROLLO DE SOFTWARE

Hay un gran número de factores que repercuten en la persona que trabaja dentro de un entorno de desarrollo de software. Los cambios en el sistema operativo disponible, el lenguaje de programación, la organización del proyecto, el formato de las especificaciones, el tipo de terminal, o los estandard establecidos para los diferentes aspectos del ciclo de vida de un proyecto pueden repercutir tanto en el trabajador como en la cantidad de trabajo que puede realizar.

La productividad, como una medida cuantitativa de la cantidad de trabajo que puede ser realizada por una persona, se puede alterar de distintas maneras; por ejemplo, enseñando a todos los implicados en el trabajo a escribir a máquina. Esto podría tener un mayor impacto en la productividad que el de introducir unas nuevas herramientas software o técnicas de diseño.

La productividad por si sola no nos dice mucho, ya que no tiene en consideración la calidad del producto. Por ejemplo, los trabajadores en una planta de ensamblaje de automóviles pueden producir 60 coches por hora, pero esta medida no es útil si un cuarto de los coches requieren trabajo adicional para corregir problemas surgidos en la etapa del ensamblaje. Igual ocurre en el desarrollo de software: el objetivo es establecer un entorno que no sólo mejore la productividad del que desarrolla el software, sino que también genere la creación de mejores productos.

El corazón del entorno es la metodología de desarrollo de software, que establece los pasos a dar con sus procedimientos y técnicas para la producción de software. La metodología generalmente consta de una secuencia de pasos, combinando procedimientos de gestión, métodos técnicos, y soporte automatizado para producir productos

software.

La relación entre una metodología y el entorno de desarrollo se muestra en la figura 2.2. Los procedimientos de gestión determinan la naturaleza del soporte automatizado que se va a proveer, en términos del equipo informático que se va emplear y de los lenguajes y herramientas que soportarán el desarrollo de software. Los procedimientos de gestión también coordinan y guían las técnicas que los informáticos emplean.

La gestión se realiza con más efectividad si se usan técnicas que provean productos intermedios, tales como documentos de especificación y de diseño. El soporte automatizado mejora la productividad del equipo de desarrollo, al mismo tiempo que valida subproductos que se van obteniendo de forma automática, evitando así el error humano y consiguientemente incrementando la calidad final.

La organización de desarrollo de software debe seleccionar entre un vasto número de posibilidades y combinaciones de métodos de gestión, técnicas de desarrollo, y soporte automatizado para crear y desarrollar la metodología de desarrollo de software.

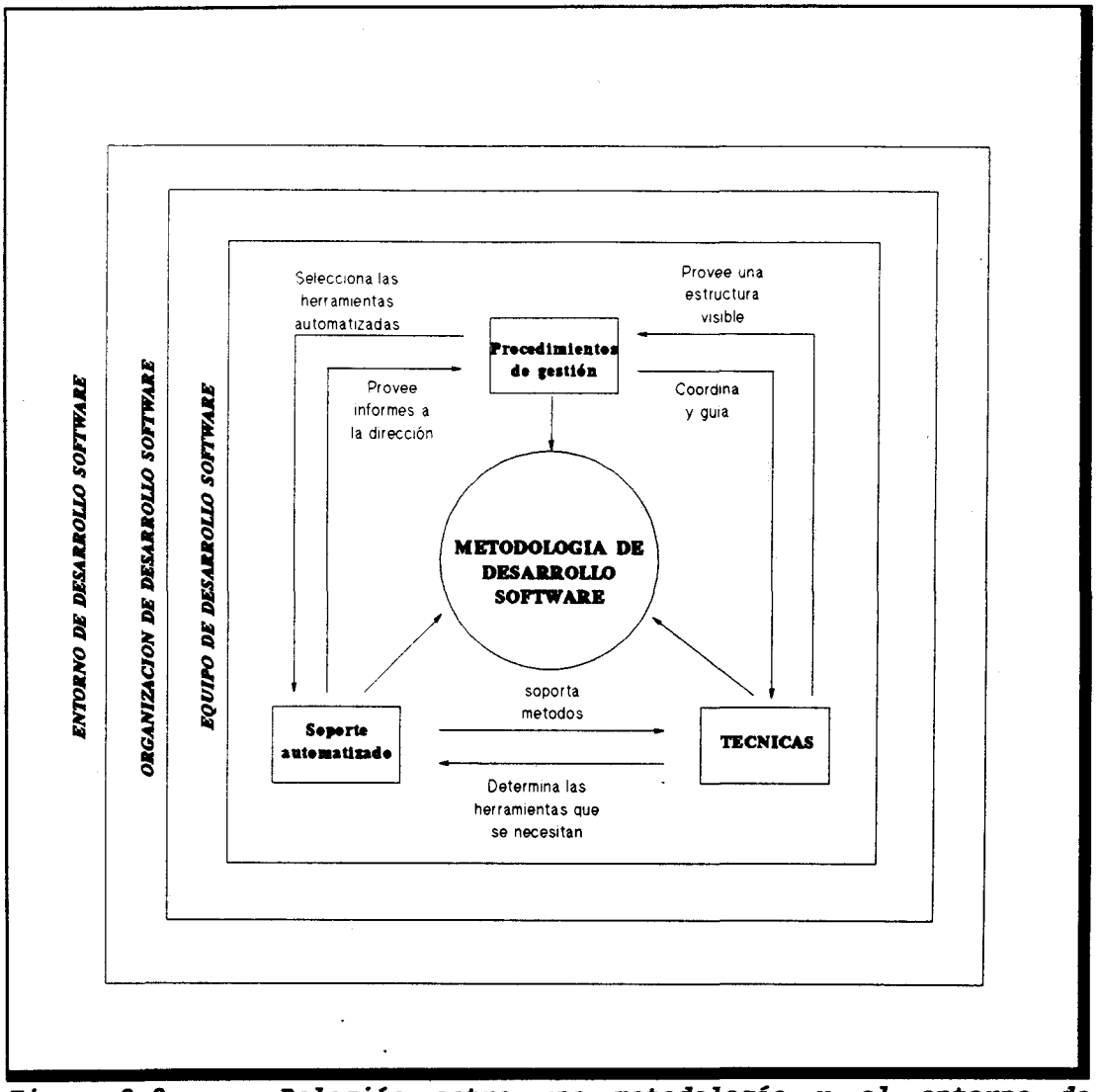


Figura 2.2 ***Relación entre una metodología y el entorno de desarrollo***

La metodología también está influenciada por muchas otras consideraciones, entre ellas: el tamaño y estructura de la organización que desarrolla el software y de las aplicaciones que se van a desarrollar, el número de veces que se van a usar, la importancia de estas aplicaciones, y las necesidades planificadas para el mantenimiento y

modificación.

Debido al gran número de posibilidades y grandes variaciones en usos y aplicaciones del software, no es razonable creer que distintas organizaciones usen la misma metodología de desarrollo de software. Incluso grupos que sigan una metodología recomendada para el desarrollo de software deberán modificarla para adaptarla a su organización y a sus proyectos. Además, las metodologías irán cambiando constantemente para reflejar nuevos desarrollos en hardware y software, así como nuevas estructuras de dirección y de personal. De hecho este aspecto de actualización es uno de los principios básicos de la ingeniería del software [BOE85].

Un ejemplo de distintas metodologías se puede observar en la producción de automóviles, la cual, como en el desarrollo de software, está regida no sólo por aspectos técnicos de elaboración y ensamblaje, sino también por la naturaleza del personal. Volvo adoptó una organización de equipo para la producción de coches, así como para mejorar la satisfacción del trabajador; averiguaron que sus trabajadores podían trabajar conjuntamente de una manera efectiva, compartir responsabilidades, y variar tareas. Bayerische Motor Werke (BMW) averiguó que tal organización

no podía ser aplicada a su entorno dada la dificultad de comunicación entre sus trabajadores a causa de las diferencias del lenguaje. Tanto Volvo como BMW producen coches aproximadamente con la misma tasa y calidad, con los mismos rangos de precios; pero diferencias en su personal les hacen emplear distintas metodologías.

2.3.- CARACTERISTICAS DESEABLES DE UNA METODOLOGIA DE DESARROLLO DE SOFTWARE

Las características deseables de una metodología para el desarrollo de software incluye las siguientes:

1.- La metodología debería cubrir el ciclo de desarrollo de software entero. De poco vale, relativamente, tener una metodología para diseño de software si no hay un procedimiento sistemático para producir la especificación usada por el diseño y/o el programa ejecutable que debe ser creado a partir del diseño. Por esto, una metodología debe asistir al informático en cada fase del ciclo de desarrollo. Ello garantizará normalmente la integración de todo el proceso productivo.

no podía ser aplicada a su entorno dada la dificultad de comunicación entre sus trabajadores a causa de las diferencias del lenguaje. Tanto Volvo como BMW producen coches aproximadamente con la misma tasa y calidad, con los mismos rangos de precios; pero diferencias en su personal les hacen emplear distintas metodologías.

2.2.1.- CARACTERISTICAS DESEABLES DE UNA METODOLOGIA DE DESARROLLO DE SOFTWARE

Las características deseables de una metodología para el desarrollo de software incluye las siguientes:

1.- La metodología debería cubrir el ciclo de desarrollo de software entero. De poco vale, relativamente, tener una metodología para diseño de software si no hay un procedimiento sistemático para producir la especificación usada por el diseño y/o el programa ejecutable que debe ser creado a partir del diseño. Por esto, una metodología debe asistir al informático en cada fase del ciclo de desarrollo. Ello garantizará normalmente la integración de todo el proceso productivo.

2.- La metodología debería integrar las transiciones entre las distintas fases del ciclo de desarrollo. Cuando un informático está trabajando en una fase determinada de un proyecto (distinta a la de análisis de requerimientos), es importante que se pueda referir a la fase previa y fusionar el trabajo. En la fase de diseño, por ejemplo, se debe asegurar que la arquitectura del sistema software cubra todas las funciones especificadas; debería ser capaz identificar los módulos software que completen los requisitos específicos del sistema. Durante la implantación, sería fácil establecer una correspondencia entre módulos en el diseño del sistema y unidades de programa, y entre los objetos de datos lógicos de la fase de diseño y los objetos de datos físicos en el programa. Es importante destacar que uno se pueda mover no sólo hacia adelante -hacia la fase siguiente- en el ciclo de vida, sino también hacia atrás -hacia la fase previa-, de tal manera que se pueda comprobar el trabajo realizado y se puedan realizar correcciones. Esta aproximación en fases al desarrollo de software hace que la información perdida en una fase determinada, generalmente quede perdida para siempre, con un impacto en el sistema resultante. Por ejemplo, si un analista deja de documentar un

requisito, éste no aparecerá en la especificación. Quizás en la prueba de aceptación, o durante la operación del sistema, aparecerá el fallo, y el sistema tendrá que ser modificado. De ahí la importancia de una validación formal de cada fase antes de pasar a la fase siguiente.

3.- La metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo. La exactitud del sistema implica muchos asuntos, incluyendo la correspondencia entre el sistema y sus especificaciones, así como que el sistema cumple con las necesidades del usuario. En consecuencia, la metodología no debe preocuparse únicamente de técnicas para validar el sistema completo, sino que también debe prestar atención para obtener la descripción más completa y consistente de las necesidades del usuario durante las fases iniciales del proyecto. Por ejemplo, los métodos usados para análisis y especificación del sistema deberían colaborar a acabar con el problema del entendimiento entre los informáticos, los usuarios, y otras partes implicadas, y hacer posible la trazabilidad hacia atrás -hacia los requisitos y la especificación- más tarde. Ello implica la existencia

de una comunicación entre usuario y técnico amigable y sencilla, exenta de consideraciones técnicas.

4.- La metodología debe soportar la organización de desarrollo de software. Debe ser posible gestionar a los informáticos, y éstos deben ser capaces de trabajar conjuntamente. Este requisito implica la necesidad de una comunicación efectiva entre analistas, programadores, y gestores, con pasos bien definidos para realizar progresos visibles durante la actividad de desarrollo. Los productos intermedios generados por los métodos y las técnicas, tales como un diseño detallado o un plan de pruebas de aceptación, deben poder ser revisados por la organización para que el progreso pueda ser medido de una manera efectiva y garantizada su calidad.

5.- La metodología debe poder emplearse en un entorno amplio de proyectos software. Serán necesarias metodologías diferentes para distintos entornos de sistemas y para distintas estructuras organizativas. Una organización deberá adoptar una metodología que sea útil para un número grande de sistemas que se vayan a construir. Es de poco sentido desarrollar una metodología para cada nuevo sistema.

6.- La metodología se debe de poder enseñar. Incluso en una organización simple, serán muchas las personas que la van a utilizar: no sólo aquellos que están cuando la metodología es adoptada por primera vez, sino también aquellos que se incorporan a la organización más tarde. Cada persona debe entender las técnicas específicas de la metodología, los procedimientos organizativos y de gestión que la hacen efectiva, las herramientas automatizadas que soportan la metodología, y las motivaciones que subyacen en la metodología. Debido a que la enseñanza de la ingeniería de software no está todavía muy avanzada, mucha de esta responsabilidad recae sobre la organización de desarrollo.

7.- La metodología debe estar soportada por herramientas automatizadas que mejoren la productividad tanto del informático en particular como la del equipo de desarrollo en general. Esta colección de herramientas, y la manera en la cual son usadas, constituye lo que ha sido llamado "entornos de programación", ya que la mayoría de las herramientas automatizadas están enfocadas a las fases de codificación e implantación del desarrollo de software.

8.- La metodología debería soportar la evolución eventual del sistema. Normalmente durante su tiempo de vida los sistema tienen muchas versiones, pudiendo durar ocho o diez años, o incluso más. Surgen nuevos requisitos debidos a los cambios de tecnología, necesidades de usuario, y estos requisitos cambiantes deben ser reflejados en el sistema modificado. La metodología de desarrollo puede asistir a esta actividad evolutiva suministrando una documentación del sistema interno y externo fiel y un buen sistema software estructurado que sea fácilmente comprensible y modificado por aquellos que realizan cambios al sistema.

En este sentido, para ayudar en el mantenimiento de los sistemas no estructurados, están surgiendo herramientas "Reverse Engineering" que permiten estructurar los componentes de éstos facilitando así su mantenimiento.

2.3 METODOLOGIAS ESTRUCTURADAS

2.3.1 ANALISIS ESTRUCTURADO DE SISTEMAS

Introducción.-

Dos versiones similares del Análisis Estructurado de Sistemas han sido descritas por Chris Gane y Trish Sarson [GAN79] y por Tom DeMarco [DEM79]. Primero pasamos a exponer la versión de DeMarco y posteriormente detallaremos las diferencias con la de Gane y Sarson.

Versión DeMarco.-

Destaca la diferencia entre los términos de análisis y análisis estructurado y entre especificación funcional y especificación estructurada.

De una manera muy breve y concisa, el análisis estructurado según DeMarco es el uso de éstas técnicas:

-Diagramas de Flujo de Datos

-Diccionario de Datos

-Especificaciones de proceso

-Tablas de Decisión

para construir el documento objetivo de la etapa de

análisis que es la especificación estructurada.

La especificación estructurada es un nuevo tipo de especificación funcional. Antes, las especificaciones funcionales entraban inmediatamente en detalles, mientras que en las especificaciones estructuradas se presenta inicialmente un gráfico que representa a todo el sistema que se va a estudiar. Empleando los principios de abstracción y jerarquización presentados en el capítulo anterior, se va pasando de lo abstracto a los detalles del sistema.

Fases y etapas.-

El Ciclo de Desarrollo de un proyecto informático según DeMarco se presenta en la figura 2.3.

El cambio más importante introducido en el Ciclo de Desarrollo es el documento de salida de la fase de análisis: la especificación estructurada, ya comentada en el punto anterior.

Las etapas de la fase de Análisis son:

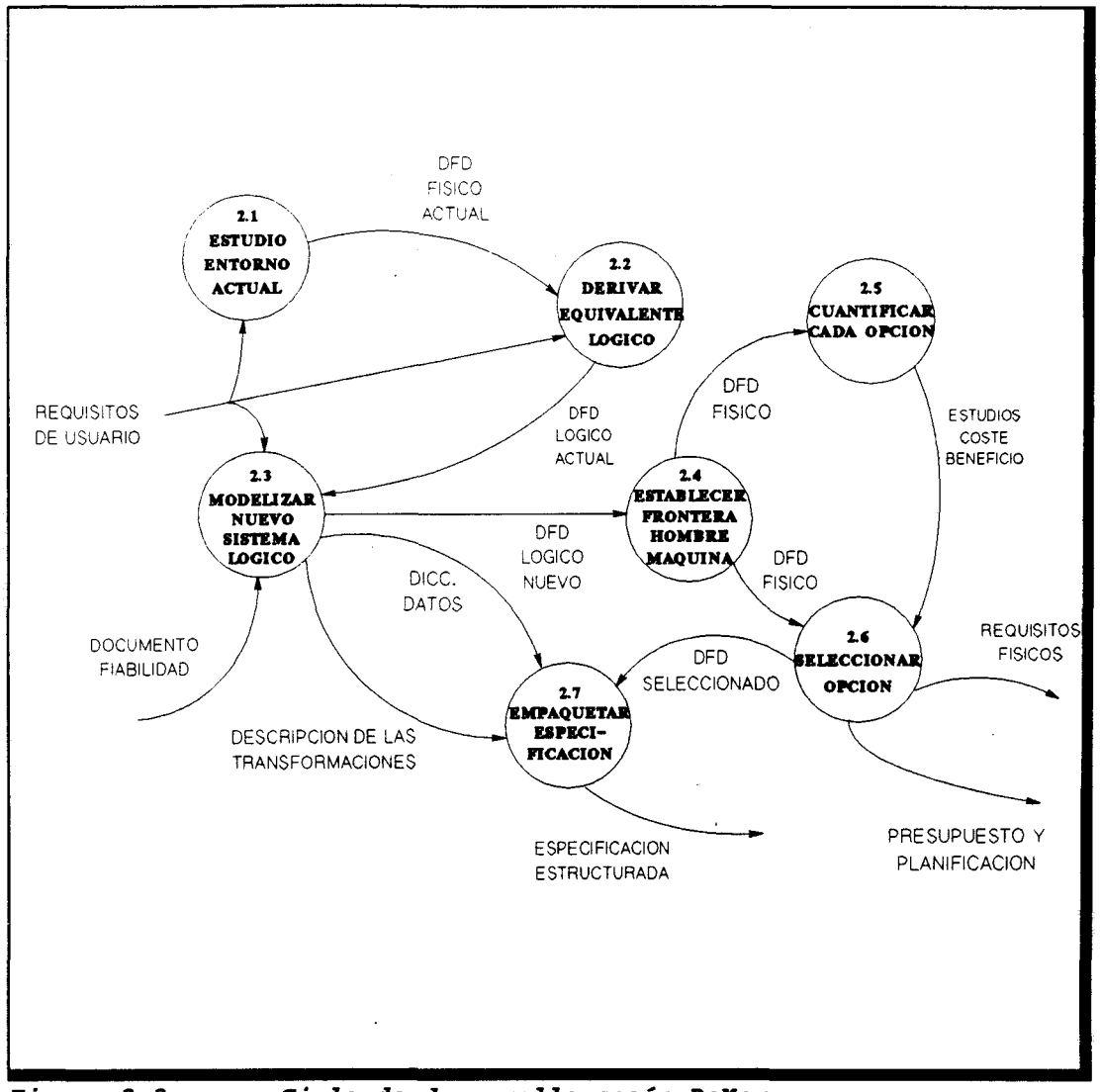


Figura 2.3 ***Ciclo de desarrollo según DeMarco***

1.- Estudio del entorno físico actual

El objetivo de esta etapa es el estudio completo de las áreas de usuario afectadas; es decir, documentar cómo se hacen actualmente las cosas en el entorno del usuario. Se muestran en esta etapa los procedimientos tanto manuales

o automatizados existentes. Se va a trabajar conjuntamente con los usuarios, y como el objetivo en esta etapa es el de realizar un modelo verificable del entorno actual, se emplearán los términos del usuario para que así lo pueda entender y validar.

Como en la representación resultante habrá elementos físicos, al producto resultante de esta etapa se le llama Diagrama de Flujos de Datos Físico Actual. Las tareas a realizar son:

- determinación del área que se va estudiar
- identificación de los usuarios afectados
- entrevistas con los usuarios
- realización de los diagramas de flujo de datos
- recolección de ejemplos de tipos de datos, ficheros e informes para completar los diagramas de flujo de datos
- revisión en equipo con los usuarios para verificar la exactitud del modelo
- publicación de la documentación resultante

2.- Derivación del equivalente lógico del entorno actual

El objetivo de esta etapa es transformar el modelo del

entorno actual en lógico. Para ello se va a partir de los diagramas de flujo de datos físico actual obtenido en la etapa anterior y se va a realizar una tarea de "limpieza", en la cual se va a eliminar todos los elementos físicos que servían como puntos de comprobación para los usuarios, reemplazándolos por su equivalente lógico.

El modelo lógico, como el modelo físico, se representa como un conjunto de niveles jerarquizados de diagramas de flujo de datos. La diferencia entre los dos modelos es que el modelo físico muestra los detalles físicos de cómo los usuarios particulares hacen actualmente las cosas, mientras que en el modelo lógico se representa gráficamente lo que se hace en un nivel abstracto, separando los procedimientos particulares de hacer las cosas de los procedimientos fundamentales que se realizan.

La salida de esta etapa son los diagramas de flujo de datos lógico actual, revisados y verificados por los usuarios.

3.- Derivación del nuevo entorno lógico

Hasta ahora no se han tenido en cuenta en las modificaciones de métodos de trabajo nuevas áreas de

negocio,etc. incluidas en el Documento de Factibilidad. Por tanto el objetivo de esta etapa es el de construir un modelo del sistema que se va a instalar.

El modelo lógico de la etapa anterior es el punto de partida. Este modelo se modifica para incorporar los cambios para el nuevo sistema, y es representado como un conjunto de diagramas de flujo de datos que describen el sistema en distintos niveles de detalle. Este paso es de gran esfuerzo en el proceso del Análisis Estructurado.

En esta etapa se trabaja de un modo lógico, es decir, que se describe qué se tiene que hacer, no cómo será llevado a cabo. No se distingue entre aquellos procedimientos que serán automatizados y aquellos que serán manuales, únicamente se *declara el trabajo que se debe* realizar, las reglas que lo gobiernan y las interfaces entre los componentes.

4.- Establecimiento del límite hombre-máquina

El objetivo de esta fase es el de crear distintas alternativas de fronteras hombre-máquina. Para ello se cuestiona en esta etapa qué áreas se van o no a automatizar en el nuevo sistema.

5.- Cuantificar cada opción

Se cuantifican los costos y los beneficios asociados para cada alternativa de modelo físico obtenido en el paso anterior.

6.- Seleccionar la opción

A partir de los resultados obtenidos en la etapa anterior, se selecciona un modelo físico para especificar las funciones y los requisitos del sistema que se va a construir.

7.- Empaquetar la especificación

Agrupar los diagramas de flujo de datos físicos nuevo seleccionados junto con toda la documentación que la soporta.

Versión Gane y Sarson.-

Esta versión se la conoce también con el nombre de metodología STRADIS SDM, (Structured Analysis Design and Implementation of Information Systems, System Development Methodology). Esta metodología se ha difundido mucho en

Estados Unidos a través de cursos y consultorías realizadas principalmente por una compañía llamada MCAUTO/IST, ligada a McDonnell Douglas, que a su vez también comercializa una serie de herramientas CASE para soportar la metodología.

FASES DEL ANALISIS ESTRUCTURADO

VERSION DEMARCO	VERSION GANE Y SARSON
1.- Construir el modelo lógico actual	
2.- Construir el modelo lógico actual - Obtención del modelo lógico de datos 3.- Construir el modelo lógico del nuevo sistema a construir - Elaborar una especificación estructurada	1.- Construir el modelo lógico actual - Limitaciones del sistema actual 2.- Desarrollar un modelo lógico del nuevo sistema - Derivar objetivos para el nuevo sistema - Producir diseños físicos tentativos alternativos - Construir un modelo de datos lógico
4.- Crear familia de nuevos modelos físicos 5.- Cuantificar cada opción 6.- Seleccionar un modelo 7.- Empaquetar la especificación	3.- Seleccionar un modelo lógico 4.- Crear el nuevo modelo físico del sistema 5.- Empaquetar la especificación

Figura 2.4 ***Proceso de desarrollo de las metodologías DeMarco y Gane&Sarson***

Gane y Sarson definen un proceso de desarrollo similar. Como se ve en la figura 2.4 la principal diferencia es que Gane y Sarson incluyen una etapa de

diseño de datos en la cual los contenidos de los almacenes que aparecen en los diagramas de flujo de datos se definen probablemente en tercera forma normal. Igualmente incluyen el diseño de datos en la etapa 2 de su versión del análisis estructurado. Luego, en la etapa 3, el modelo de datos lógico se usa como entrada en el diseño físico de la base de datos. También, como se observa en la figura 2.5, Gane y Sarson utilizan un conjunto de símbolos gráficos diferentes en el empleo de la técnica de Diagramas de Flujo de Datos.

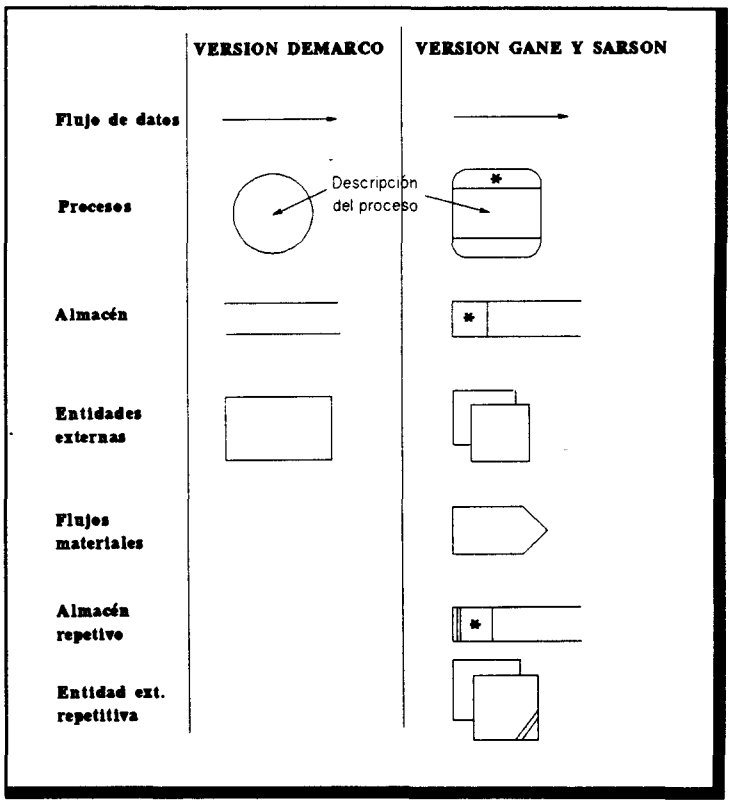


Figura 2.5 Símbolos usados en un Diagrama de Flujo de Datos

Tanto la metodología de DeMarco como la de Gane y Sarson están siendo mejoradas y refinadas, ya que sus trabajos iniciales se fundamentaron en la exposición de las técnicas que utilizaban, y se están expandiendo hacia áreas de diseño e implantación.

Ambas metodologías han sido concebidas para ser aplicadas al desarrollo de cualquier sistema de información, independientemente del tamaño o de si el sistema va a ser o no automatizado. En la práctica, sin embargo, han sido principalmente usadas y refinadas en entornos donde el sistema de información está automatizado.

Técnicas.-

Las técnicas empleadas en estas dos metodologías son:

- Diagramas de Flujos de Datos
- Diccionario de Datos
- Lenguaje Estructurado
- Tablas de Decisión

DIAGRAMA DE FLUJOS DE DATOS

Un diagrama de flujo de datos (DFD) es una representación gráfica de un sistema en forma de red, entendiendo por sistema todo el conjunto de procedimientos, ya estén mecanizados o sean manuales. Un DFD muestra todos los procedimientos y las conexiones entre los mismos procedimientos del sistema. Los diagramas de flujo de datos son construidos con cuatro elementos básicos:

- Flujo de datos: representados por vectores etiquetados, simbolizan los datos en movimiento a lo largo de todo el sistema. Sirve de conexión entre el resto de los componentes de un DFD.

- Procesos: representados por círculos (en versión DeMarco), simbolizan transformaciones de los datos que entran al círculo vía flujo de datos, en datos que salen de él, también vía flujo de datos.

- Almacén de datos: representados por dos líneas paralelas (versión DeMarco), simbolizan los datos que están durante un cierto tiempo en reposo.

- Entidades externas: representadas por rectángulos. Cualquier persona u organización, que no pertenece al sistema es una entidad externa. Estas sirven para representar la fuente y el destino de los datos del sistema.

Con los diagramas de flujo de datos se representan los cuatro modelos del sistema durante sus distintas fases de refinamiento: físico actual, lógico actual, lógico requerido, físico requerido. La figura muestra un diagrama de flujo de datos de la etapa de análisis estructurado

según la metodología DeMarco.

Los diagramas de flujo de datos proveen una descripción gráfica excelente que es fácil de entender para los usuarios. Suministran una visión clara de la funcionalidad del sistema y de los límites del sistema.

DICCIONARIO DE DATOS

El diccionario de datos, o repositorio, es el lugar donde se tienen almacenados todos los datos y todas las actividades que son requeridas en el sistema.

El diccionario de datos es creado a la vez que se va realizando los diagramas de flujos de datos durante el análisis del sistema existente. Las entradas son realizadas cada vez que se identifica un:

- flujo de datos
- componente del flujo de datos
- almacén
- proceso funcional

Se utilizan tres operadores relacionales para definir cualquier entrada en el diccionario de datos.

- secuencia, mediante el símbolo "+"
- decisión alternativa, mediante el símbolo "[/]"
- decisión en bucle, mediante el símbolo "{ }"

Ejemplos:

Secuencia: Cliente = Nombre + Dirección

Repetición: Nombre Hijos = 0{ Nombre Hijo }10

Selección: Residencia = [Casa / Apartamento / Hotel]

LENGUAJE ESTRUCTURADO

El lenguaje estructurado o seudocódigo es un lenguaje de especificación que utiliza un vocabulario limitado, el de un lenguaje natural como el inglés o castellano, y una sintaxis limitada, la sintaxis de la programación estructurada.

El vocabulario consta de:

- verbos infinitivos precisos
- términos definidos en el diccionario de datos
- ciertas palabras reservadas para formulación lógica

La sintaxis del lenguaje estructurado está limitada a estas posibilidades:

- sentencia declarativa simple
- construcciones de decisión cerradas
- construcciones de repetición cerradas

Ejemplo:

```
PARA CADA LINEA ALBARAN HACER
    SI   DESCUENTO
         MULTIPLICAR VALOR_LINEA POR 100 MENOS DESCUENTO
         Y DIVIDIR RESULTADO ENTRE 100
    CASO CONTRARIO (NO DESCUENTO)
         VALOR_LINEA NO CAMBIA
```

TABLAS DE DECISION

En los casos donde la lógica del proceso es bastante compleja para el uso de las sentencias IF del lenguaje estructurado es de gran ayuda usar tablas de decisión.

Una tabla de decisión es una expresión tabular de decisiones procedurales que están caracterizadas por un conjunto de acciones que son ejecutadas en función del estado de las condiciones.

El ingrediente básico de una tabla de decisión es la regla de decisión. Una regla de decisión describe un

conjunto de condiciones alternativas y una serie de acciones a realizarse.

Hay dos partes básicas de una tabla de decisión: la parte de condición y la parte de acción. La parte condición está formada por un conjunto de condiciones y entradas de condiciones. La parte de acción está formada por un conjunto de acciones y entradas de las acciones.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Pedido > límite de crédito	C	F	C	F	C	F	C	F	C	F	C	F	C	F	C	F
Pedido aprobación especial	C	C	F	F	C	C	F	F	C	C	F	F	C	C	F	F
Pedido < cantidad de envio	C	C	C	C	F	F	F	F	C	C	C	C	F	F	F	F
Envio aprobación especial	C	C	C	C	C	C	C	C	F	F	F	F	F	F	F	F
Rechazar crédito			X				X			X				X		
Rechazar envio								X	X	X	X					
Procesar pedido	X	X		X	X	X		X				X	X		X	X

Figura 2.6 ***Tabla de Decisión***

2.3.2 TECNICAS DE ANALISIS Y DISEÑO ESTRUCTURADO (SADT)

Introducción.-

SADT nació en EEUU en 1976 gracias a la sociedad americana Softech en colaboración con la ITT, cuyo principal artífice fue D.T.Ross [ROS77]. El método es llamado Análisis Estructurado y Técnica de Diseño (SADT). SADT abrió un nuevo camino en las áreas de análisis de problemas, definición de requisitos y especificación funcional, ya que permite una expresión rigurosa de ideas de alto nivel que previamente habían parecido demasiado indeterminadas para tratar técnicamente. Este método ha adquirido un renombre internacional, lo mismo a nivel de grandes grupos industriales (Thomson, Philips, Aérospatiale, Toshiba,...) como de organismos nacionales o internacionales (Agencia Espacial Europea, CNES,...). SADT fue introducido en Francia, por IGL Technology en 1982, formando a una gran cantidad de ingenieros para la especificación de sistemas, lo que ha hecho que sea uno de los standards de la informática industrial.

Fases y Etapas.-

El SADT puede ser usado efectivamente para todas o cualquier fase del desarrollo del sistema.

Los siguientes modelos se pueden producir:

* Modelo del Entorno Actual.
* Modelo de Operaciones Actuales.
* Modelo del Nuevo Sistema:
    - Modelo de Requerimientos.
    - Modelo Funcional.
    - Modelo de Diseño.
        - Mecanismos, Modelos Funcionales y de Diseño.
* Modelo del Plan de Test.
* Nuevo Modelo del Entorno.
* Nuevo Modelo de Operaciones.
* Modelo de Conversión.
* Modelo de Desarrollo del Sistema.

* <u>Modelo del Entorno Actual</u>

- Muestra cómo opera sistema actual haciendo énfasis en las partes en donde el nuevo sistema lo reemplazará.

- La comunicación con el resto del sistema total es parte del contexto limitado del nuevo sistema.
- Muestra las funciones de la parte del sistema total que va a ser reemplazada, dando luz en las funciones del nuevo sistema.

* Modelo de Operaciones Actuales

- Muestra cómo la gente (operadores, usuarios) se relaciona con el actual sistema, con énfasis en las partes que pueden ser cambiadas con el nuevo sistema.

Los dos modelos anteriores se harán entrevistando a los expertos, usuarios, operadores y estudiando la documentación del sistema.

* Modelo del Nuevo Sistema

- Modelo de Requerimientos: muestra todos los requerimientos del nuevo sistema.
- Modelo Funcional: expresa los requerimientos funcionales y sus implicaciones.
- Modelo de Diseño: captura el diseño que cumple todos los requerimientos.
- Modelos de Mecanismos: los dos anteriores aislan partes separables de acuerdo con el principio del

SADT "dividir y conquistar".

* Modelo del Plan de Test

- Muestra cómo el nuevo sistema será probado.
- Incluye planes para:
  + verificar que satisface necesidades reales,
  + descubrir y modificar errores,
  + afinar el sistema.

* Modelo del Entorno

- Muestra cómo el nuevo sistema incide en el entorno.
- Es usualmente una modificación del modelo actual del entorno.
- El nuevo sistema (Modelo de Diseño) incide en este modelo.

* Nuevo Modelo de Operaciones

- Muestra cómo la gente incide con el sistema total cuando el nuevo sistema es instalado.
- Es una modificación del modelo actual de operaciones.

* Modelo de Conversión

- Muestra cómo el nuevo sistema será traído "ON-LINE".
- El modelo de conversión muestra un camino ordenado

de plantear el paso entre:

+ viejo entorno y nuevo,

+ viejas operaciones y nuevas.

* Modelo de desarrollo del Sistema

- Muestra rigurosamente y en detalle el proceso completo.
- Muestra los pasos en el proyecto, etapas, la gente y sus funciones.

Técnicas.-

Las técnicas que se emplean para la realización de todos los modelos en SADT son:

- Diagramas de actividades
- Diagramas de datos

DIAGRAMA DE ACTIVIDADES Y DE DATOS

Al diagrama de actividades se les conoce también por "actigram" y al diagrama de datos por "datagram". En un actigram los nodos denotan actividades y los arcos especifican flujos de datos entre actividades. Los actigrams son similares a los diagramas de flujos de datos. Los datagrams especifican datos en los nodos y actividades

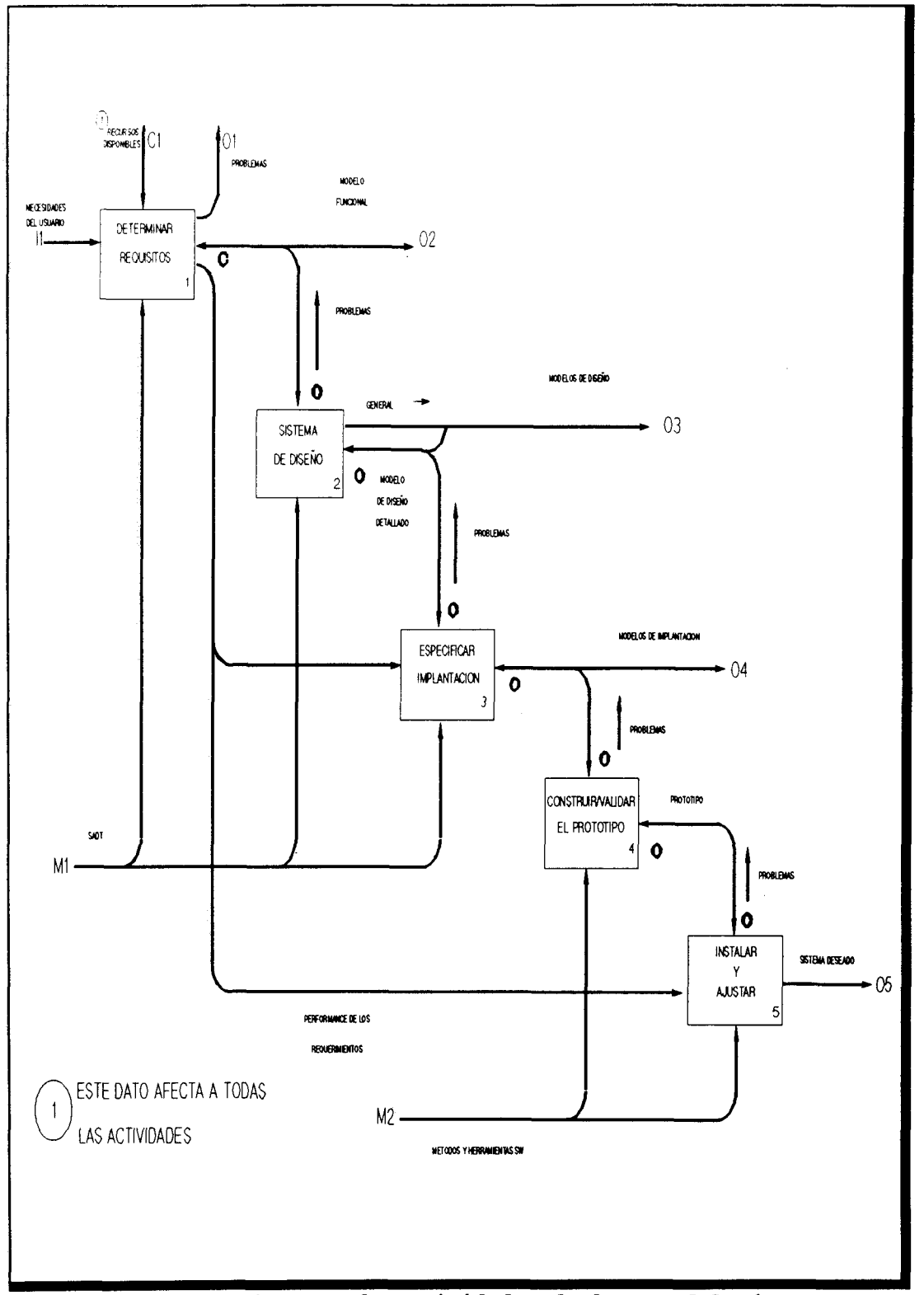


Figura 2.7 ***Diagrama de actividades de la metodología SADT***

en los arcos. Por esto, los actigrams y los datagrams son duales. En la práctica, los diagramas de actividades se utilizan con más frecuencia que los de datos; sin embargo, los de datos son importantes por lo menos por dos razones: para indicar todas las actividades afectadas por un dato y para verificar la integridad y consistencia de un modelo SADT mediante la construcción de diagramas de datos a partir de un conjunto de diagramas de actividades.

Las figura muestran los formatos de los nodos de los actigrams y datagrams. A un nodo se le puede conectar cuatro distintos tipos de arcos. Los arcos que entran por el lado izquierdo del nodo son entradas y los arcos que salen por el lado derecho son salidas. Los arcos que entran por la parte superior son de control y los que entran por la parte inferior especifican mecanismos.

Las salidas proporcionan entradas y controles de otros nodos. Las salidas de algunos nodos son las salidas del sistema hacia el ambiente externo.

Cada salida debe conectarse a otro nodo o hacia el exterior. De la misma manera, las entradas y controles deben venir de otros nodos o del exterior. Las entradas, controles y salidas pueden estar conectados a nodos en

otras páginas en un conjunto de diagramas.

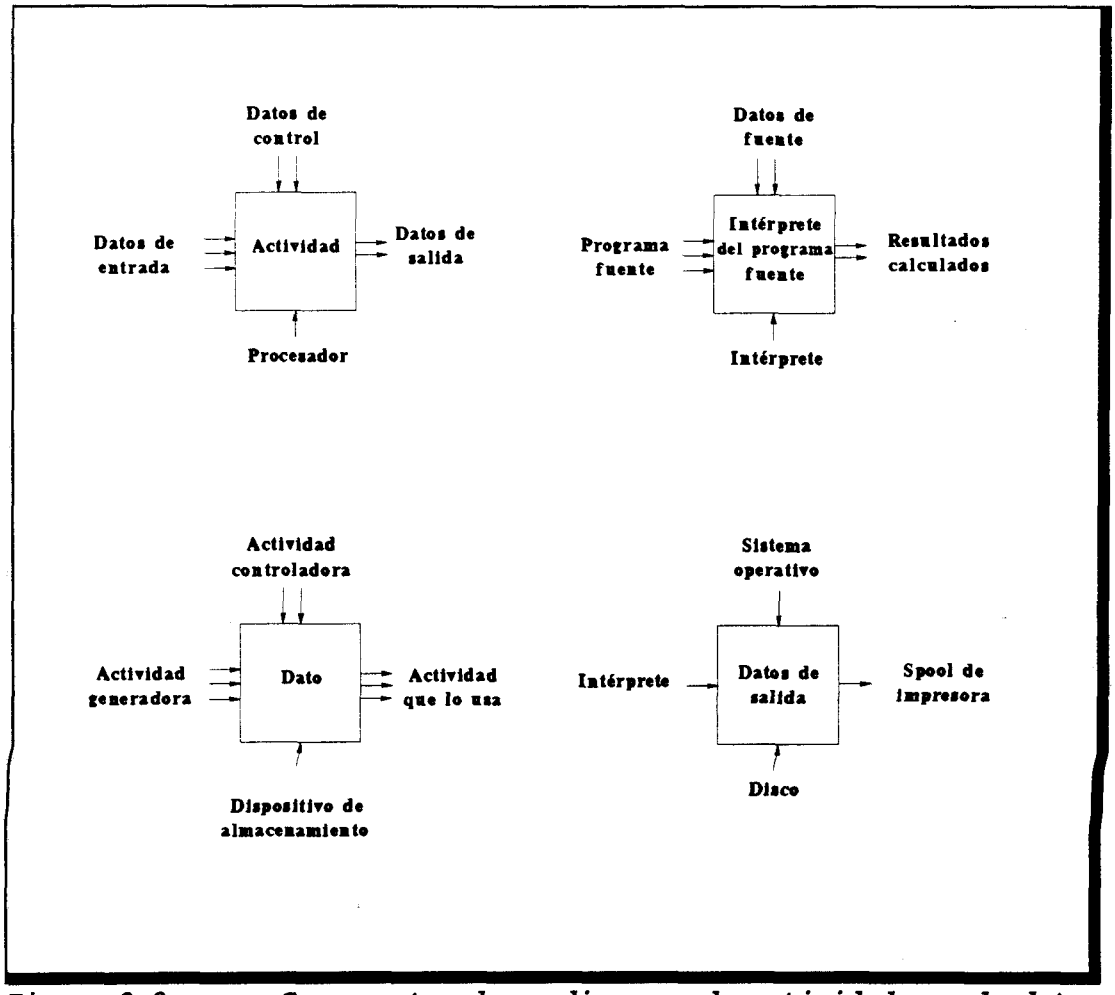


Figura 2.8 ***Componentes de un diagrama de actividades y de datos***

En un diagrama de actividades las entradas y las salidas son flujos de datos y los mecanismos son procesadores (mecánicos o humanos). Los controles son datos utilizados pero no se modifican en la actividad.

En un diagrama de datos la entrada es la actividad que crea el dato y la salida es la actividad que lo emplea. Los mecanismos en los diagramas de datos son los dispositivos utilizados para almacenar las representaciones de los datos. El control en los datagrams controla las condiciones bajo las cuales el nodo será activado.

Tanto en los actigrams como en los datagrams los controles provienen del ambiente externo o son salidas de otros nodos.

2.3.3 METODOLOGIA ESTRUCTURADA DE ANALISIS Y DISEÑO DE SISTEMAS (SSADM)

Introducción.-

La metodología SSADM ha sido aceptada como estandard por muchas organizaciones de gran envergadura en el mundo entero y por el gobierno británico. En 1980, la empresa consultora Learmonth y Burchett Management Systems (LBMS) fué invitada a trabajar con la Central Computer and Telecommunications Agency (CCTA), la cual es responsable, entre otras funciones, de la formación de informática y telecomunicaciones para el Servicio Civil del Reino Unido para realizar un proyecto conjunto con el objetivo de desarrollar un método estandard de análisis y diseño para el gobierno británico. El resultado fue SSADM, un estandard obligatorio en la actualidad para todos los proyectos del gobierno británico. Se puede encontrar una descripción de la metodología en [DOW88].

Fases y etapas.-

SSADM se divide en seis fases (ver figura 2.9). Cada una de ellas representa las actividades de una fase bien definida del ciclo de desarrollo. Cada fase tiene su propio

conjunto de objetivos definido cuidadosamente.

Los objetivos de cada fase son:

1.- Análisis

Construir un modelo lógico del sistema actual. Documentar los problemas del sistema actual y los requerimientos para el nuevo sistema.

2.- Especificación de requerimientos

Construir un modelo lógico del sistema requerido junto con documentación detallada.

3.- Selección de opción del sistema

Identificar y documentar los requerimientos opcionales para el nuevo sistema.

4.- Diseño lógico de datos

Completar una especificación de diseño lógico de datos detallada.

5.- Diseño lógico de procesos

Completar un conjunto de diseños lógicos de procesos detallados.

6.- Diseño físico

Trasladar el diseño lógico de datos a las especificaciones de ficheros o bases de datos y los diseños lógicos de procesos a las especificaciones de programa.

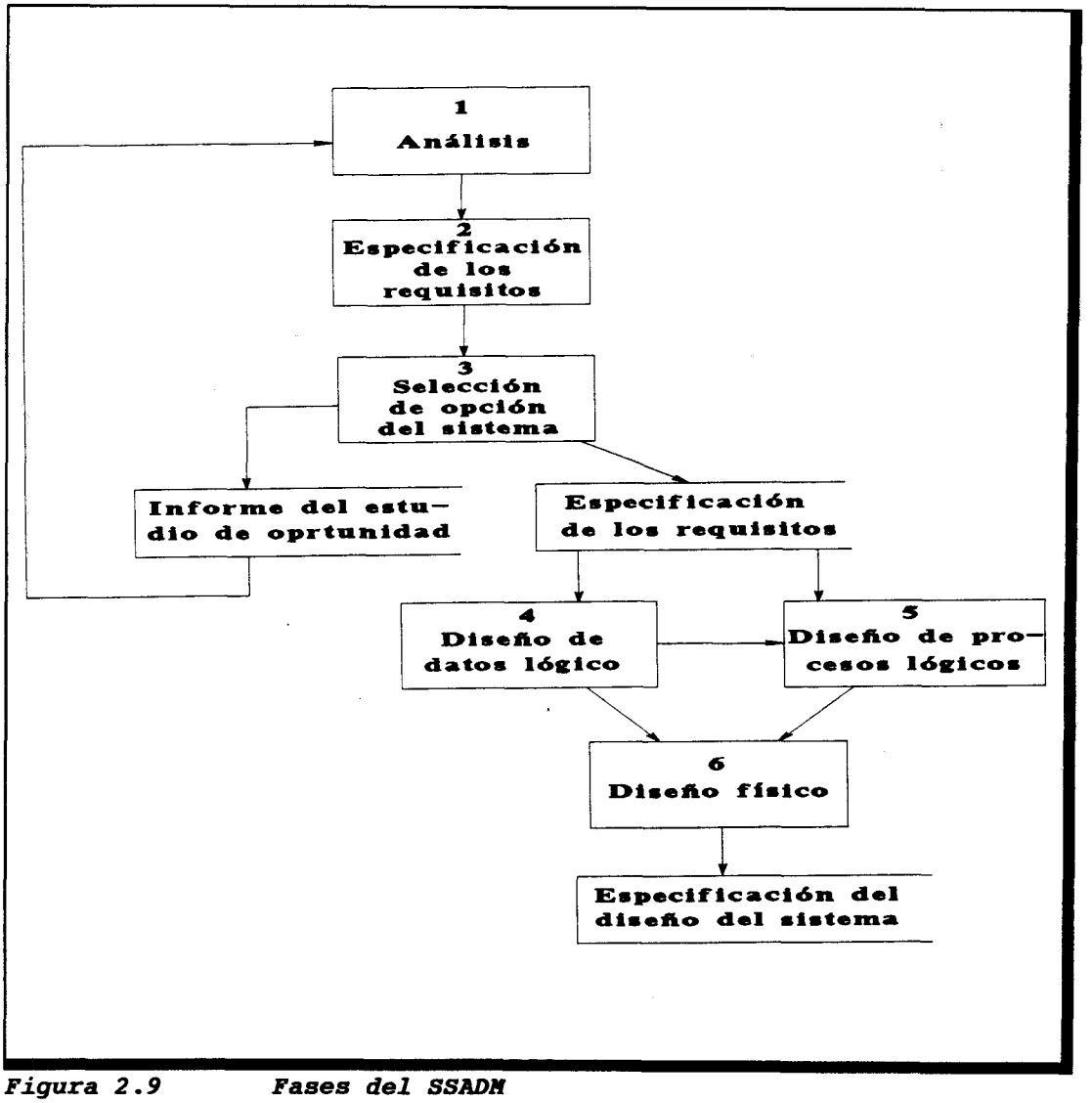


Figura 2.9 *Fases del SSADM*

Cada fase comprende varias tareas que proporcionan una aproximación estructurada que nos encamina al cumplimiento de los objetivos de la misma.

SSADM es una metodología para el análisis y diseño de sistemas. SSADM comienza organizando las notas de investigación y concluye produciendo unas especificaciones detalladas de diseño para los implantadores.

Esta metodología no incluye fases para investigación o implantación. La fase de investigación de un proyecto se apoya en técnicas comunes a muchas situaciones de gestión, como por ejemplo entrevistas. SSADM puede comenzar detrás de una investigación detallada.

La implantación comienza con la programación y la prueba de programas. La fase 6 de SSADM está hecha a medida para asegurar que el diseño estructurado conecte con los métodos de programación existentes y sobre todo aportándole una documentación detallada.

Técnicas.-

Las técnicas utilizadas en la metodología SSADM son:

- Diagramas de flujo de datos

- Modelos de entidad
- Matriz de entidades y funciones
- Historia de la vida de la entidad
- Esquema de proceso lógico

DIAGRAMAS DE FLUJOS DE DATOS

El diagrama de flujo de datos empleado por SSADM es semánticamente idéntico al de Gane y Sarson, y al de DeMarco. Emplea un conjunto de símbolos algo más rico, pero no permite el desdoblamiento de flujos de datos creando por esta razón innecesarios flujos de datos y haciendo menor la legibilidad del diagrama.

MODELOS DE ENTIDAD

Los modelos de entidad proporcionan una visión del sistema de las estructuras de datos y relaciones entre los datos en el sistema. Todos los sistemas tienen un modelo de entidad subyacente que permanece prácticamente invariable en el tiempo. El modelo de entidad refleja la lógica de los datos del sistema, no la implantación física.

Hay muchos ejemplos donde diferentes departamentos de la misma organización realizan funciones idénticas de muy

distintas maneras. Sus respectivos diagramas de flujos de datos físicos actuales deberían reflejar las diferencias. Sus modelos de entidad deberían ser muy similares, ya que los datos y las relaciones de datos requeridos para realizar la función son los mismos.

Los modelos de entidad son lógicos, no físicos; representan grupos de datos lógicos, llamados entidades, y las relaciones entre las entidades. Cliente y pedido son ejemplos de entidades; un encargo es una relación entre la entidad cliente y la entidad pedido. La figura 2.10 muestra un modelo de entidad.

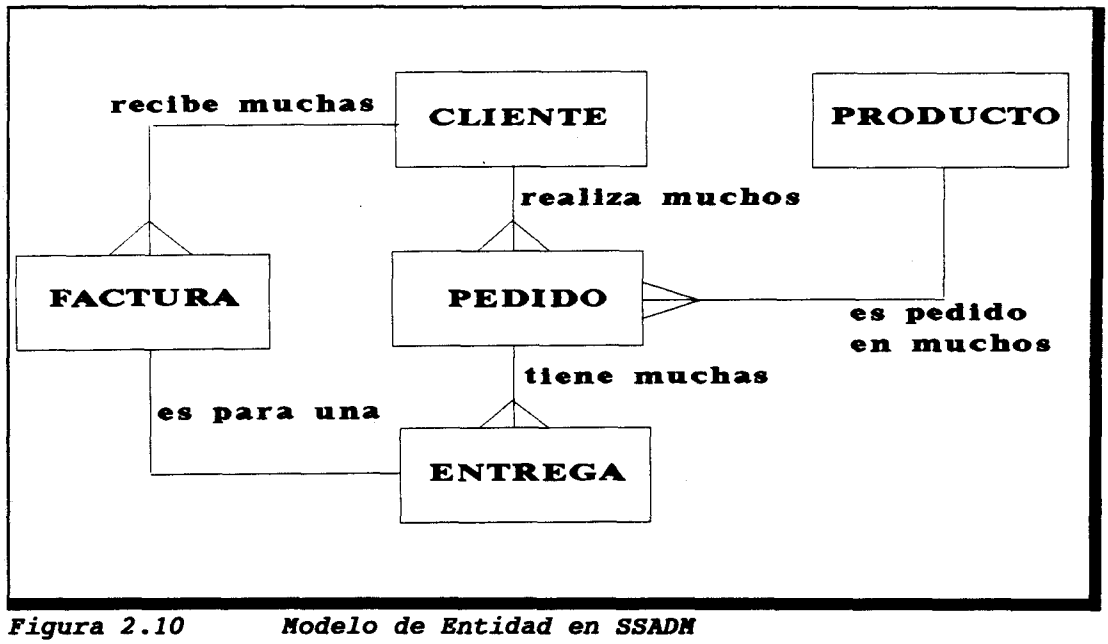


Figura 2.10 Modelo de Entidad en SSADM

Los modelos de entidad son usados en las fases 1, 2 y

4. Progresan desde los primeros intentos de entender las subyacentes estructuras y relaciones de los datos (durante la fase 1), hacia la construcción del modelo de entidad resultante (durante la fase 4). Los modelos de entidad proporcionan una excelente representación gráfica de las estructuras de datos y relaciones genéricas. Proporcionan asimismo una visión clara de la estructura lógica de los datos dentro de los límites de interés y permiten al analista crear modelos de datos sin considerar su forma física. El uso del modelo de entidad ofrece una visión del sistema independiente del proceso actual; es una visión de todo el sistema, no una visión descompuesta por funciones.

La técnica del modelo de entidad complementa los diagramas de flujos de datos proporcionando la visión del sistema que falta sobre las estructuras de datos y sus relaciones.

MATRIZ DE ENTIDADES Y FUNCIONES

La matriz de entidades y funciones proporciona una visión dinámica del sistema. Traza el efecto en el tiempo de las funciones sobre las entidades. Los diagramas de flujo de datos proporcionan una visión del comportamiento del sistema y los modelos de datos una visión de la

estructura de los datos, sin referencia al tiempo. En esta matriz se detalla la secuencia de eventos, no su temporización real.

Las funciones forman las columnas y las entidades las filas de la matriz de entidades y funciones. El esquema de la matriz se crea listando todas las entidades y todas las funciones del sistema.

El nuevo modelo lógico proporciona la fuente para la lista de funciones. Cada función primitiva, o sea, las funciones que no son descompuestas posteriormente, se convierte en una columna. La retención de los números de función del diagrama de flujo de datos en la matriz proporciona unas referencias cruzadas útiles. El modelo de la entidad proporciona la lista de entidades.

Las entradas dentro de la matriz pueden contener una o más de las siguientes letras:

espacio : la función no tiene efecto en la entidad
I : la función inserta la entidad en la base de datos
M : la función modifica la entidad
L : la función lee la entidad
B : la función borra la entidad

Nombre de Función / Nombre de Entidad	1.1 Entrada de pedido	1.2 Producir acep-tación pedido	1.3 Registrar envio	1.4 Producir factura	2.1 Crear nueva cuenta cliente
Cliente	R	R			I
Pedido	I	M	M	D	
Producto			M		
Envio			I	D	
Factura				I	

Figura 2.11 ***Matriz de Entidades y Funciones***

Las entradas se hacen por referencia a los DFD. Todos los flujos de datos entre funciones y almacenes de datos se convierten en una o más entradas en la matriz.

Cada fila de la matriz muestra la vida de una entidad. Muestra qué funciones insertan la entidad, qué funciones

la leen o modifican, y qué funciones la borran. Cada fila debería comprender al menos una I (inserción), una L (lectura) o M (modificación) y una B (borrado); la aplicación de esta regla nos permite descubrir errores cometidos en los DFD, y como consecuencia de ello pueden aparecer nuevas funciones y/o desaparecer algunas de ellas.

HISTORIA DE LAS ENTIDADES

Cada fila de la matriz de entidades y funciones proporciona una lista de funciones con su efecto sobre una entidad. La matriz de entidades y funciones no puede mostrar la secuencia de funciones para cada entidad, ni puede mostrar el efecto de situaciones anormales. Las historias de entidad se trazan para representar la secuencia de funciones en la vida de una entidad y para representar e identificar el efecto de las situaciones anormales. Cada situación anormal requiere una función que procese tal situación.

Se usan cuatro elementos basados en las Redes de Petri para representar la historia de la entidad:

- la elipse se usa como símbolo de comienzo y fin. Se anota con el nombre de la entidad.

- el cuadrado se usa para las funciones. El nombre de la función en el DFD se escribe en la caja de la esquina inferior derecha del cuadrado. Su efecto sobre la entidad se escribe en la caja de la esquina inferior izquierda.
- el círculo se usa para mostrar el estado de la entidad. Los círculos simplemente son numerados.
- las flechas indican la transacción de un estado a otro por la acción de una función.

Las funciones que insertan, modifican o borran entidades se muestran en la historia de la entidad, porque modifican el estado de la entidad. Las funciones que sólo leen no modifican el estado de la entidad y por tanto no aparecen en la historia de la entidad.

En la historia de una entidad se puede representar las estructuras estandard de secuencia, selección e iterativas y además se pueden representar la estructura de concurrencia con la cual podemos mostrar vidas concurrentes.

En la historia de una entidad hay que tener en cuenta una vida anormal de tal entidad. En este estudio pueden identificarse funciones nuevas. Estas funciones nuevas

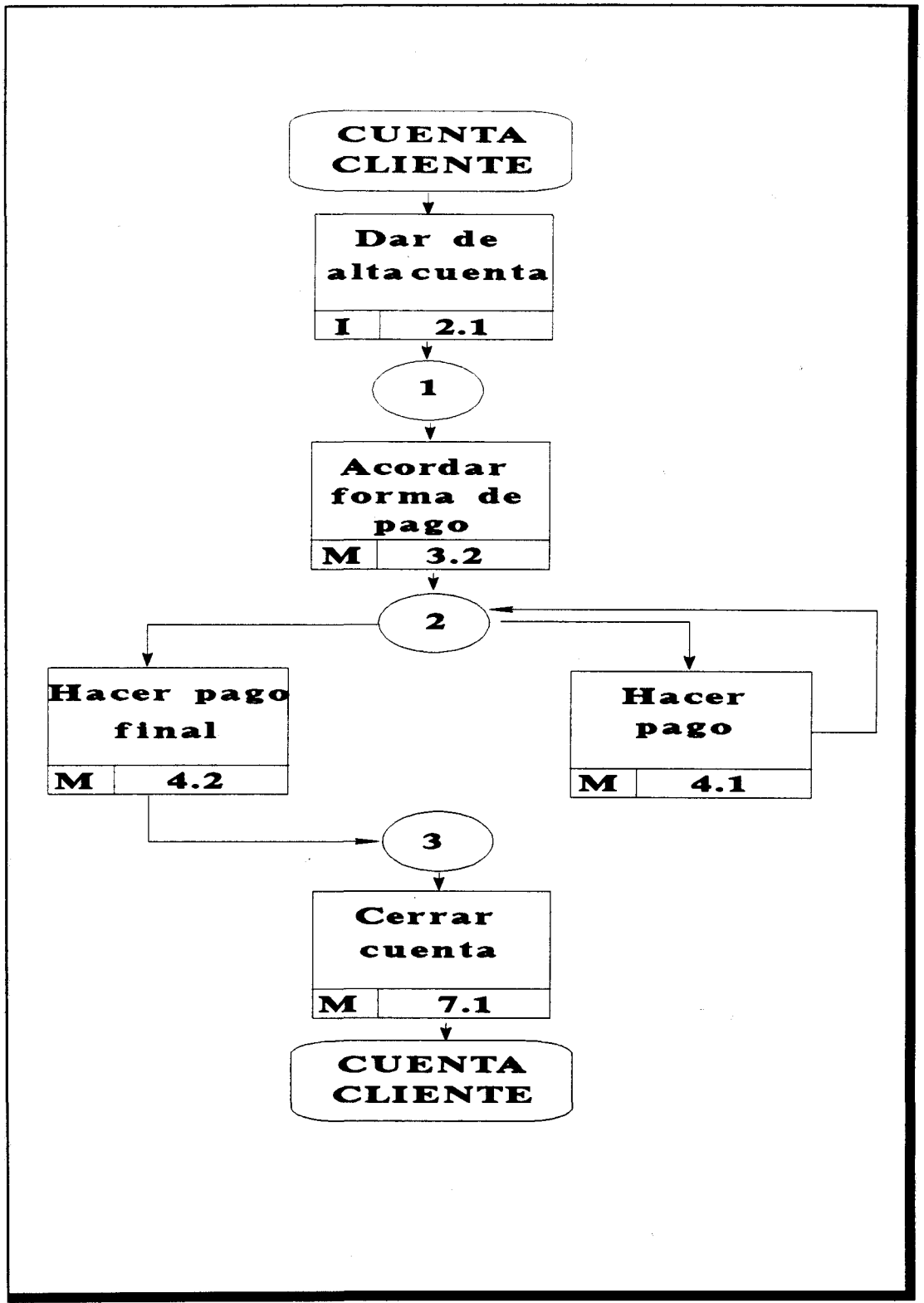


Figura 2.12 ***Historia de la Entidad***

deben ser registradas en el nuevo modelo lógico.

La construcción de una historia de entidad asegura que cada entidad tiene una vida completa. Cada ocurrencia de la entidad en la base de datos debe estar en una etapa apropiada de su vida, siendo representada por el indicador de estado o vector de estado.

Además, cada vida completa tiene un comienzo, una vida normal, una posible vida anormal, y una serie de funciones que siempre conducen al fin de la vida. Las ocurrencias de entidad se insertan, se comportan adecuadamente y son eventualmente borradas de la base de datos. Se debe asegurar al diseñador de la base de datos que las funciones especificadas por el sistema actúan sobre las entidades en la base de datos de una forma prescrita y controlada.

Cada ocurrencia de la entidad tendrá asociado el estado con esa ocurrencia. Esto limita instantáneamente el número de los procesos válidos que pueden tener un efecto sobre la entidad. Cada especificación de proceso comprobará primero el estado de la entidad antes de la ejecución, estableciendo el nuevo estado después de la terminación afortunada del proceso.

ESQUEMA DE PROCESO LOGICO

Un esquema de proceso recoge todas las operaciones necesarias para ejecutar un proceso. Las entidades afectadas se obtienen de la matriz de entidades y funciones. El estado anterior de la ocurrencia de la entidad para la que es válida la función se establece a partir de la historia de la entidad, que también proporciona el estado resultante de la terminación de la función con éxito.

CREACION FACTURA

ENTIDAD	EFECTO	ESTADO ENTRADA	ESTADO SALIDA
Línea de pedido	M	3	3
Envio	M	2	3
Factura	I	–	1

Figura 2.13 ***Esquema de Proceso Lógico***

El esquema de proceso se puede construir con éxito fácilmente y comprende el nombre de la entidad, el efecto

sobre la ocurrencia de la entidad y los estados de entrada y de salida válidos (ver figura 2.13). El esquema de proceso puede ser mejorado para incluir una descripción del procesamiento para cada operación requerida en la realización del proceso. Cada esquema de proceso incluirá una o más caja de función de los diagramas de flujo de datos. Un esquema de proceso completo se muestra en la figura 2.13.

2.3.4 WARD & MELLOR

Introducción.-

La metodología definida por Paul T. Ward y Stephen J. Mellor [WAR85] está orientada para ser empleada en la definición de un sistema de tiempo real. La definición toma la forma de un modelo riguroso que describe la funcionalidad esencial; es decir, lo que debe hacer el sistema y qué datos se deben almacenar, así como los requisitos de tiempo y datos, y los detalles de la implantación propuesta.

Fases y etapas.-

La evolución de un sistema según esta metodología pasa por realizar tres actividades:

- Construir el modelo esencial: describe el comportamiento del sistema requerido.

- Construir el modelo de implantación: describe la organización de la tecnología automatizada que va a dar cuerpo al comportamiento requerido.

- Construir el sistema: engloba el modelo de implantación en hardware y software.

Exponemos únicamente el modelo esencial porque las otras dos actividades sobrepasan el alcance de esta investigación.

El proceso de construcción del modelo esencial consta de dos partes:

- un modelo enfocado a definir con qué debe interaccionar el sistema, llamado Modelo del Entorno.

- un modelo que describe el comportamiento requerido del sistema, llamado Modelo de Comportamiento.

Ambos modelos están libres de la implantación, es decir, no hay afirmaciones realizadas acerca de la tecnología que va ser utilizada para implantar el sistema. Ambos modelos, sin embargo, deben tener en cuenta la tecnología de implantación usada en el entorno (la cual, por definición, no es parte del sistema), incluyendo aquellos asuntos asociados con los fallos o errores.

El Modelo del Entorno es una descripción del entorno en el cual el sistema opera. Este modelo tiene dos partes:

- una frontera entre el sistema y el entorno, mostrando las interfaces entre las dos partes. A esta descripción se le llama Diagrama de Contexto.

- y una descripción de los eventos que suceden en el entorno al que el sistema debe responder, llamada Lista de Eventos.

El Modelo de Comportamiento es una descripción del comportamiento requerido del sistema. Este modelo también tiene dos partes que están conectadas:

- la Transformación del Esquema, que representa gráficamente un conjunto de procesos que actúan cuando suceden unos eventos en el entorno.

- Esquema de Datos con sus especificaciones, que representa gráficamente el conjunto de información que el sistema debe tener para poder funcionar.

Técnicas.-

Las técnicas empleadas en la metodología Ward y Mellor son:

- Esquema de Transformación
- Diagramas de transición de estados
- Especificación de la transformación de los datos
- Verificación del esquema de transformación
- Esquema de datos

ESQUEMA DE TRANSFORMACION

El esquema de transformación es idéntico al diagrama de flujo de datos de DeMarco y al de Gane y Sarson en cuanto a la vista de datos; pero el esquema de transformación incorpora además una vista de eventos al diagrama de flujo de datos. Esta vista de eventos contiene los siguientes elementos:

- flujos de eventos: Son simples señales que indican que algo ha ocurrido o dan una orden. Se representan mediante flechas con líneas a trazos.

- transformaciones de control: Es una transformación o proceso (en vocabulario de DeMarco) que acepta

únicamente flujos de eventos como entradas y produce únicamente flujos de eventos. Se representan mediante círculos con líneas a trazos.

- almacenes de control: que se usan para almacenar ocurrencias de flujos de eventos, y se le representa mediante un par de líneas a trazos paralelos.

DIAGRAMAS DE TRANSICION DE ESTADOS

Los diagramas de transición de estados se emplean para especificar cada transformación de control. Los componentes de este diagrama son:

- Estados: Representados mediante un rectángulo. Simbolizan un modo de comportamiento que se puede observar desde el exterior.

- Transiciones: Representados mediante una flecha. Simbolizan el movimiento de un estado a otro.

- Condiciones: Aparecen junto al estado encima de una línea. Hace que el sistema realice una transición.

- Acciones: Aparecen junto al estado debajo de una línea. Son realizadas cuando ocurre una transición.

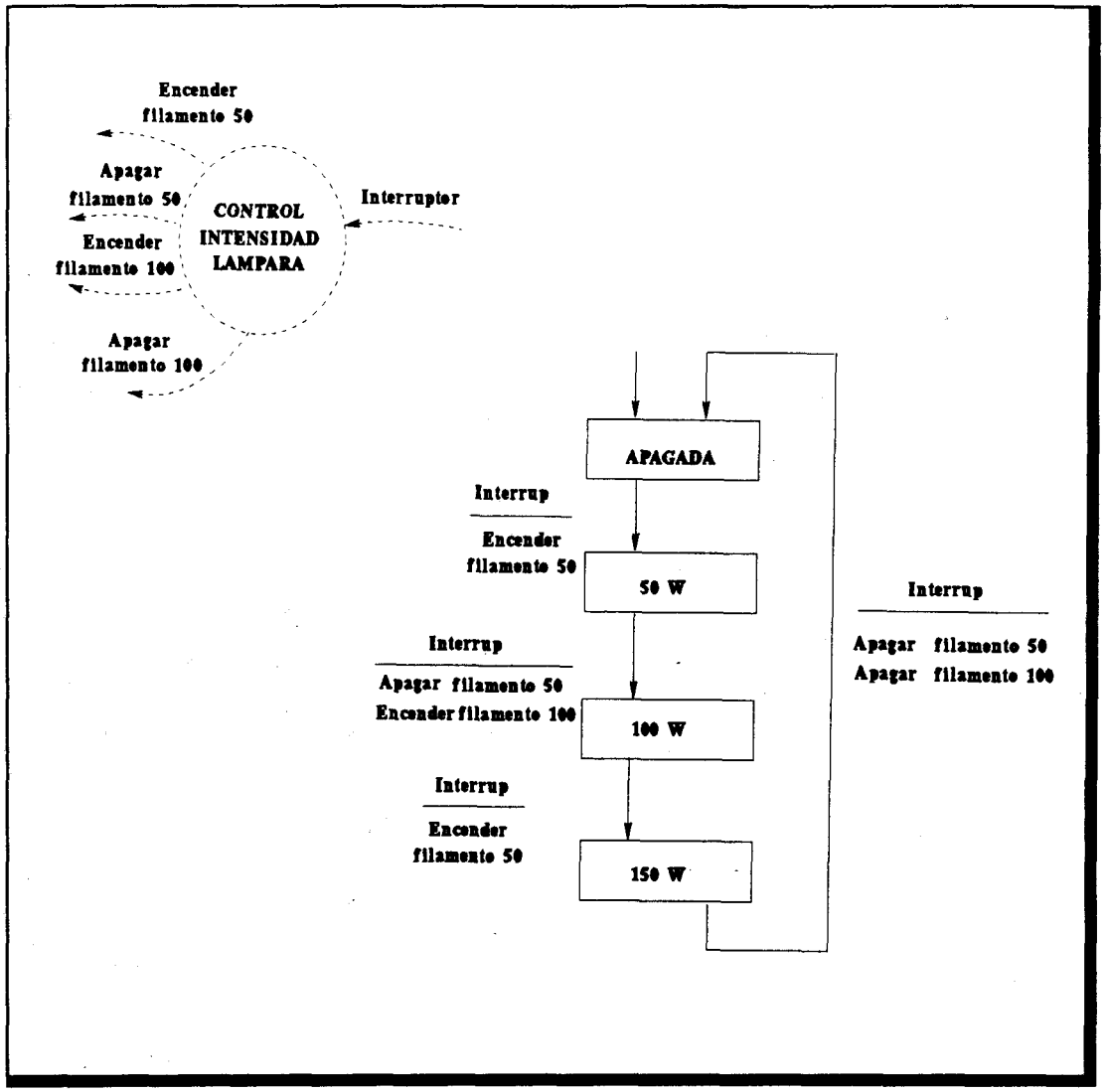


Figura 2.14 ***Diagrama de Transición de Estados y su representación en el Esquema de Transformación***

ESPECIFICACIÓN DE LAS TRANSFORMACIONES DE DATOS

Para describir la función que debe llevarse a cabo por

un proceso emplea la técnica de pre/post condición. Esta técnica es muy adecuada porque describe la función que debe realizarse sin decir mucho del algoritmo o procedimiento que se usará. Las precondiciones describen todo aquello que debe ser cierto para que el proceso arranque. Similarmente las postcondiciones describen lo que debe ser cierto para cuando el proceso finalice.

Ejemplo: ACUMULAR ROTACIONES

Precondición 1:

ROTACIONES ACUMULADAS = 0

Postcondición 1:

ROTACIONES ACUMULADAS = Integral de 0 a T de TASA ROTACION * dt

VERIFICACION DEL ESQUEMA DE TRANSFORMACION

El objetivo de tal verificación es mostrar que el modelo general del sistema es correcto; es decir, que el sistema produce las salidas correctas cualitativamente, dado un conjunto de entradas. Para ello se realiza una ejecución a nivel esquemático de todo o parte del esquema de transformación. Esto se hace a través de un escenario

que consta de varios pasos, cada uno de los cuales describe una parte del mundo real a simular y una especificación de las situaciones de las señales requeridas.

ESQUEMA DE DATOS

El esquema de datos emplea la notación y conceptos básicos del diagrama de Entidad-Relación de Chen [CHE76] con algunas extensiones y reglas de refinamiento propuestas por Flavin [FLA81]. La notación basica consta de:

Objeto-tipo: Representa un conjunto de entidades del mundo real que juega un determinado papel en el sistema. Gráficamente se representa mediante rectángulos.

Relación tipo: Representa una asociación de un determinado tipo de interacción entre los objetos-tipo a los que conecta. Se representa mediante rombos. Las relaciones son inherentemente multidireccionales.

Podrán existir múltiples ocurrencias tanto de un objeto-tipo como de una relación-tipo. Pueden existir dos o más relaciones entre los mismos objetos-tipo; una

relación puede conectar un objeto-tipo consigo mismo. En la figura 2.15 aparece un ejemplo de Esquema de Datos.

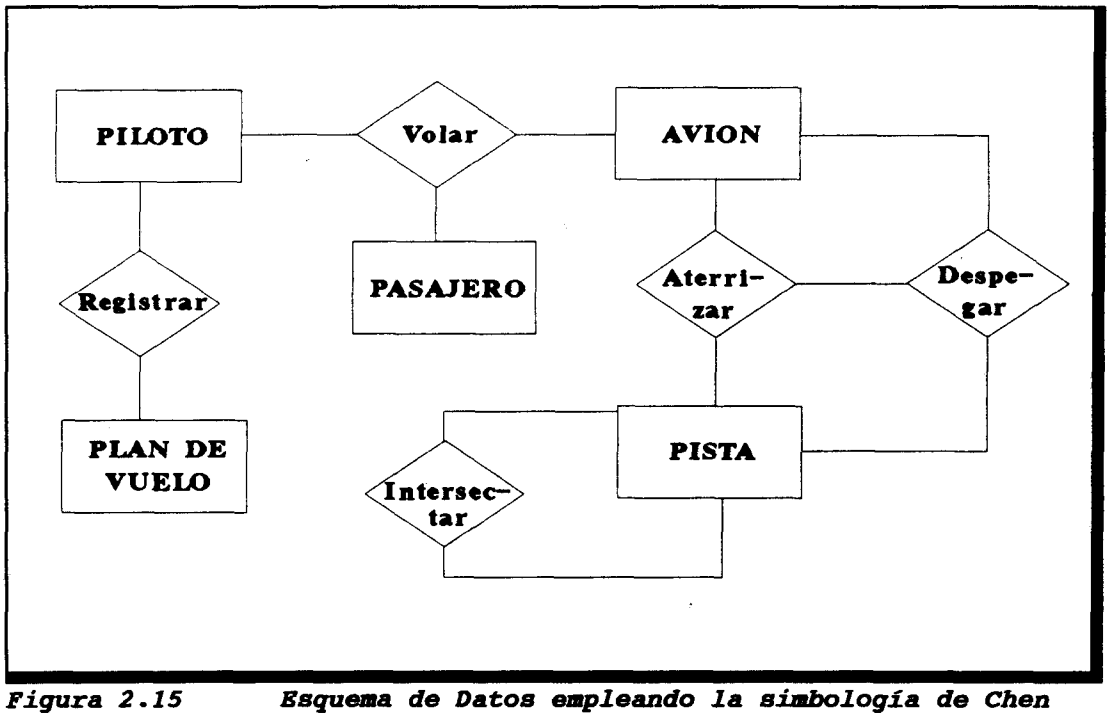


Figura 2.15 ***Esquema de Datos empleando la simbología de Chen***

2.3.5 METODOLOGIA MERISE

Introducción.-

Las bases del método Merise comenzaron a desarrollarse en el año 1972 en el seno de un pequeño equipo universitario de ingenieros en Aix-en-Provence. Este equipo fue animado desde el principio por Hubert Tardieu, concluyendo una primera versión a finales de 1976.

Colaboraron en su perfeccionamiento organismos del gobierno francés y otras universidades, siendo en la actualidad un método bastante completo en su ámbito, formalizado, simple de entender y usar.

Fases y tareas.-

En la metodología Merise se establece una separación entre el modelo de datos y el modelo de tratamientos. La separación entre un modelo y otro está bastante generalizada en los distintos métodos, pero una buena aportación de Merise es que valida ambos modelos para averiguar si son compatibles.

En un desarrollo informático se presentan distintos

tipos de problemas a solucionar, que requieren técnicas y conocimientos distintos. Merise hace una distinción entre tres tipos de problemática fundamentales, que son los siguientes:

Nivel Conceptual:

Corresponde a las finalidades de la empresa. Se describe a través de un conjunto de reglas de gestión, objetivos y limitaciones que pesan sobre la empresa. Constituye el nivel más estable y en él se encuentran por ejemplo: reglas de gestión del personal, de contabilidad, de entrega de productos acabados,etc.

Nivel Organizativo:

Corresponde a la organización adecuada a implantar en la empresa para alcanzar los objetivos considerados. Precisará los puestos de trabajo, cronología de las operaciones, elección de la automatización, integrando todas las limitaciones eventuales menos estables.

Nivel Técnico:

Corresponde a la integración de los medios técnicos necesarios para el proyecto. Se expresan en términos

de materiales o componentes software. Este nivel está sujeto a más cambios como resultado de los progresos tecnológicos.

En cada nivel se obtienen los siguientes resultados:

NIVELES	DATOS	TRATAMIENTOS
Conceptual	**Modelo Conceptual de los datos**	**Modelo Conceptual de los tratamientos**
Organizativo	**Modelo Lógico de los datos**	**Modelo Organizativo de los tratamientos**
Técnico	**Modelo Físico de los datos**	**Modelo Operativo de los tratamientos**

Figura 2.16 ***Niveles de MERISE***

Para establecer las descripciones de los datos y de los tratamientos a los distintos niveles se dispone de formalismos y técnicas específicas.

Las fases que cubre la metodología de Merise son dos:

- Estudio previo:

En el que se realizará un armazón de las soluciones funcionales y técnicas y se obtendrán los modelos de datos y tratamientos.

- Estudio detallado:

El ámbito de esta fase será el de un subdominio funcional. Se obtendrán unos estudios detallados y unos cuadernos de carga para la futura realización de los programas productos.

Técnicas

Las técnicas empleadas en MERISE son:

- Formalismo Individual
- Formalismo de Redes de Petri

FORMALISMO INDIVIDUAL

Esta técnica se emplea en la etapa conceptual de los datos, cuyo objetivo es el de obtener un modelo conceptual validado de la realidad a partir del diccionario de datos, de las limitaciones y reglas de gestión, así como de los modelos externos.

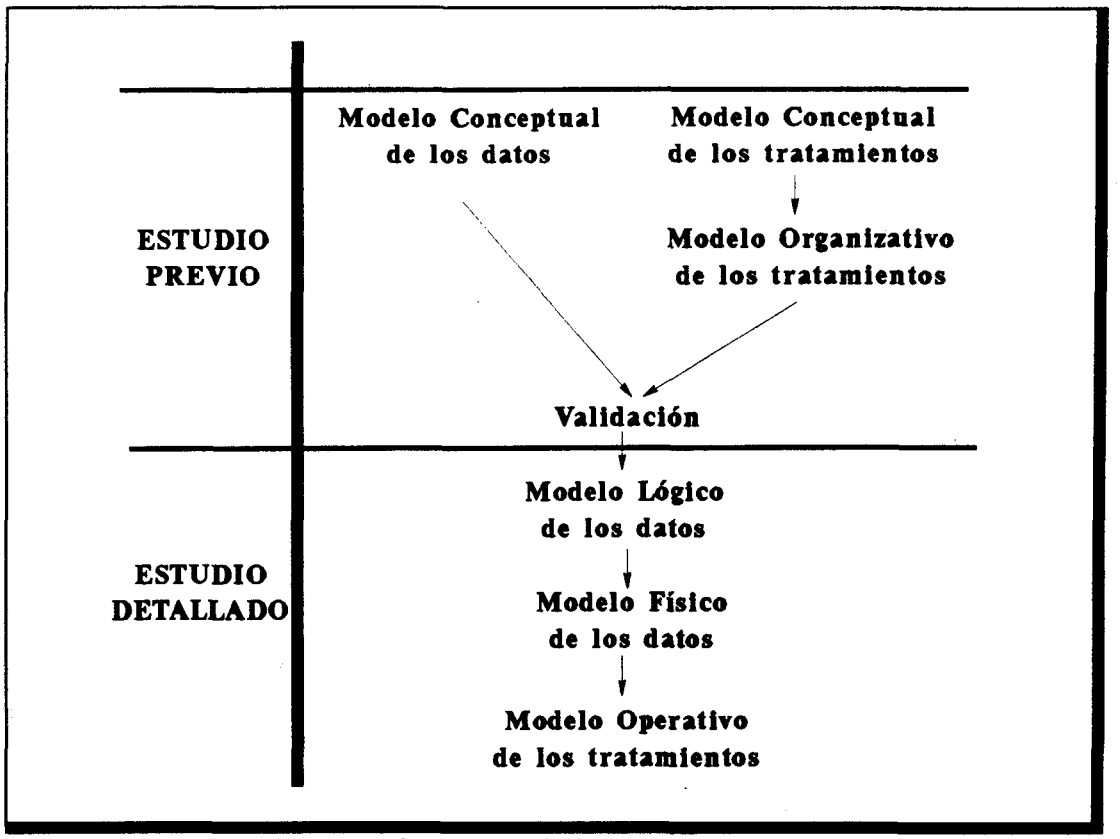


Figura 2.17 ***Secuencia de actividades en MERISE***

El formalismo individual es parecido al modelo entidad-relación [CHEN76] pero con la diferencia de que no utiliza ninguna limitación informática. Parte de cuatro conceptos:

Individuo: Objeto que tiene existencia propia para las finalidades del proyecto; por ejemplo: Producto, Cliente, etc.

Relación: Objeto que no tiene existencia propia en las finalidades del proyecto, sino que señala una conexión entre dos o más individuos; por ejemplo: propiedad de Individuo PERSONA sobre Individuo COCHE.

Atributo: Información elemental. Pertenece a un individuo o propiedad.

Cardinalidad: Expresa la participación de un individuo en una relación.

El individuo se representa mediante un rectángulo y contiene un recuadro superior con el nombre y una serie de atributos; uno de ellos identificador y el resto constituye la entidad.

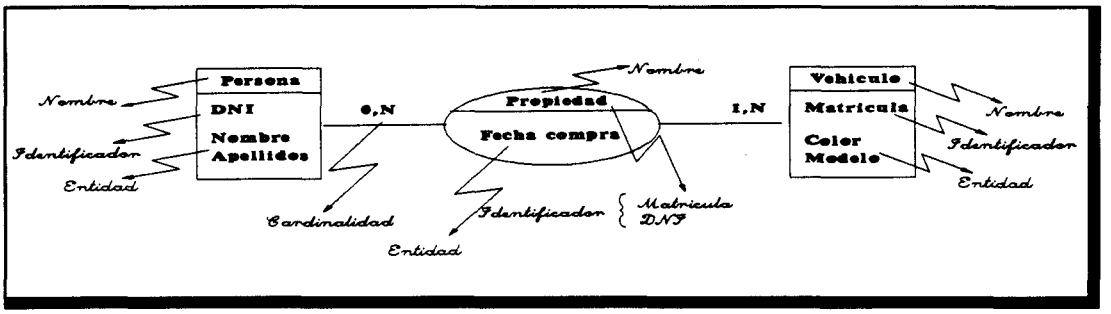


Figura 2.18 Simbología del Formalismo Individual

Una relación se representa mediante una elipse. La concatenación de los identificadores de los individuos que participan en la relación constituye el identificador de la relación. El resto de las propiedades de la relación constituye la entidad como se observa en la figura 2.18.

La cardinalidad de PERSONA en la relación PROPIEDAD indica que por cada ocurrencia de persona es propietaria de cero a n vehículos, y la cardinalidad de VEHICULO en la relación PROPIEDAD señala que por cada ocurrencia de VEHICULO ésta puede ser de propiedad de una a n personas.

FORMALISMO DE REDES DE PETRI

Esta técnica se usa para realizar el Modelo Conceptual de Tratamientos, cuyo objetivo es el de determinar las acciones llevadas por la empresa independientemente de cómo ésta haya decidido organizarlas y despreciando el "dónde", "el quién", el "cuándo" y el "cómo".

Esta técnica manipula los conceptos de:

Acontecimiento: Es el hecho de que algo se ha producido en el entorno o en el sistema de información mismo. Tiene por efecto desencadenar un

conjunto de acciones; ejemplo: LLEGADA DE UN PEDIDO.

Operación: Es una acción o conjunto de acciones a cumplir por los módulos de programas del sistema de información como reacción a un acontecimiento desencadenadas de forma ininterrumpida; ejemplo: EFECTUAR PRUEBAS DE CONOCIMIENTO EN UNA CLASE, CORREGIR Y ANOTAR ESTAS PRUEBAS.

Sincronización: Definida como precondiciones que deben satisfacerse para que una operación arranque. Es una condición booleana que traduce las reglas de gestión que deben verificar los acontecimientos para desencadenar las acciones; ejemplo: SI PETICION A SATISFACER.

La simbología que emplea se muestra en la figura 2.19. Las acciones han sido descubiertas previamente en el establecimiento de las especificaciones. Para cada una de ellas se han precisado los acontecimientos que la desencadena, las reglas de gestión conforme a las cuales se desarrolla y los resultados que produce.

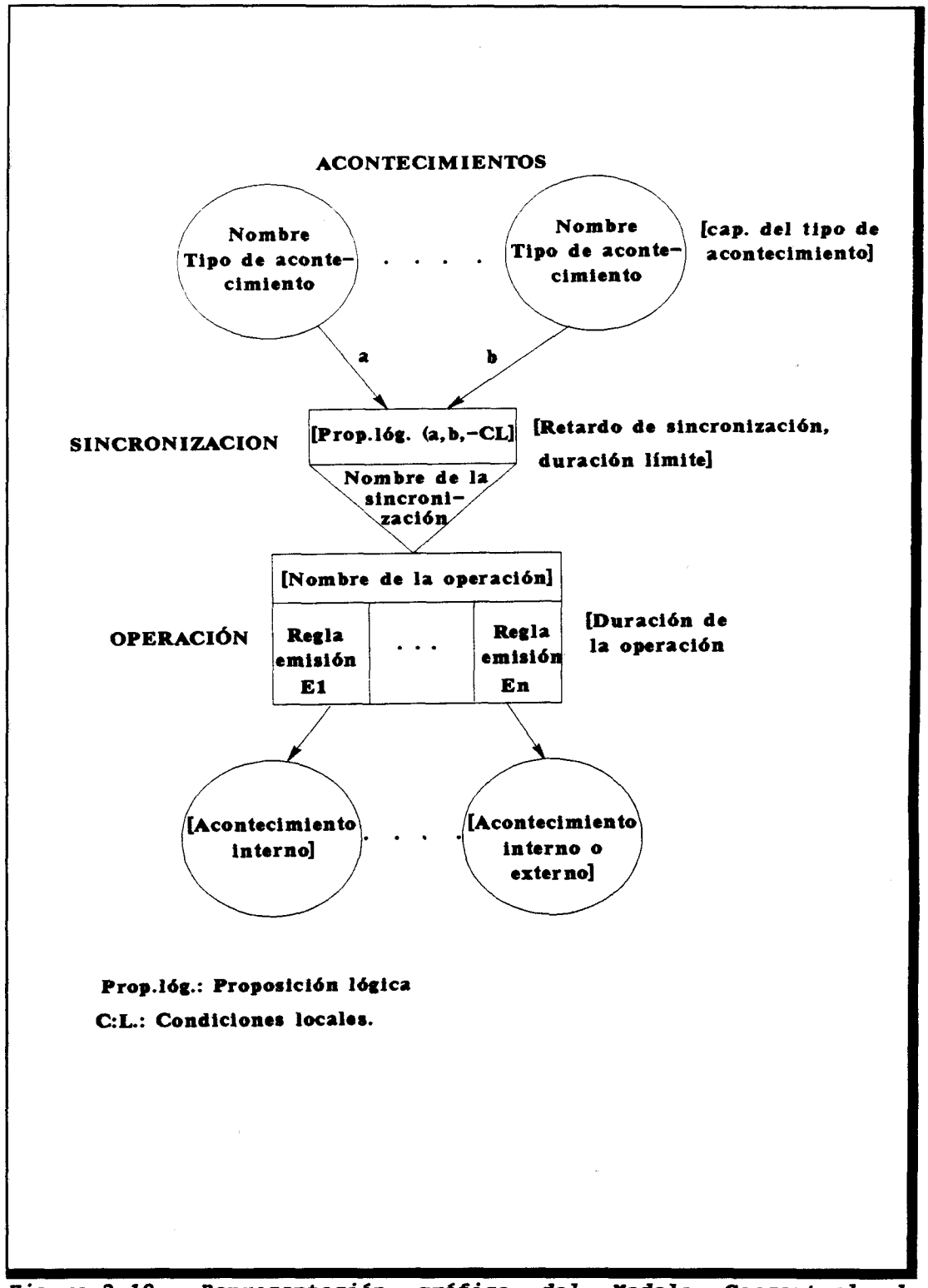


Figura 2.19 Representación gráfica del Modelo Conceptual de Tratamientos

2.3.6.- DESARROLLO DE SISTEMAS JACKSON (JSD)

Introducción.-

La metodología de diseño de programas de Michael Jackson (JSP) [JAC75] ha tenido un efecto profundo en la enseñanza y práctica de la programación de problemas de gestión. La metodología JSD [JAC83], parte de que el diseño de sistemas es una extensión de la tarea de diseñar programas, y que las mismas técnicas se pueden aplicar útilmente a ambas. Los aspectos de JSP son difundidos a través de JSD, de modo que la metodología JSD es desarrollo de su precursora. Por tanto, no tiene que ser vista como una frontera con JSP sino como una extensión de ella.

Fases y etapas.-

Un principio fundamental en JSD es que sólo cuando se ha realizado un modelo de la realidad se pasa a considerar las funciones que debe desarrollar el sistema. El sistema en sí es visto como una simulación de la realidad y las funciones se construyen sobre esta simulación.

Hacer un modelo JSD del mundo real conlleva dos tareas:

- hacer una descripción abstracta del mundo real
- hacer una realización, en el ordenador, de esa descripción.

La descripción abstracta del mundo real es simplemente una descripción en la que se consideran los aspectos relevantes y se omiten aquellos que son irrelevantes. La realización debe satisfacer la descripción abstracta para el propósito del modelo. La descripción abstracta se lleva a cabo en dos pasos, y su realización en el tercer paso, y no es hasta el cuarto paso cuando se consideran las funciones del sistema. Las funciones del sistema deben expresarse en términos cuyo significado depende del punto de vista del usuario sobre la realidad. Si el punto de vista de la realidad cambia, las funciones deben cambiar, pero no al contrario; la función puede cambiar y sin embargo la realidad subyacente permanecer constante. En un paso siguiente se describen las realidades dinámicas; es decir, aquellas en las cuales ocurren eventos en un cierto orden en el mundo real. Estos eventos pueden variar o ser fijos pero siempre son de vital importancia en la realidad. El modelo dinámico construido está compuesto por un conjunto de procesos secuenciales, que son simples programas estructurados, y son secuenciales porque en ellos no hay ninguna noción de concurrencia, paralelismo o

multitarea. Sólo una cosa puede ocurrir en un instante dentro de un proceso secuencial. El trabajo del diseñador del sistema, en la etapa de implantación, consiste en considerar que en el conjunto del modelo podrán ocurrir varias cosas al mismo tiempo, ya que habrá muchos procesos en el modelo y habrá que transformar estos procesos a una forma que pueda ejecutarse eficientemente.

En la figura 2.20 se muestra las seis etapas del proceso de desarrollo JSD. A continuación se hace una breve descripción de cada una de estas etapas.

1.- Entidad-Acción

Aquí se define el área del mundo real que concierne al sistema a desarrollar, estableciendo una lista de entidades y las acciones que éstas ejecutan o sufren, haciendo además una descripción informal de cada acción.

2.- Estructura de las entidades

Se ordenan en el tiempo las acciones de una entidad. Ocasionalmente se puede encontrar que una entidad puede ejecutar o sufrir sus acciones en cualquier orden, sin embargo éstas están siempre sometidas a unas restricciones en cuanto a su orden.

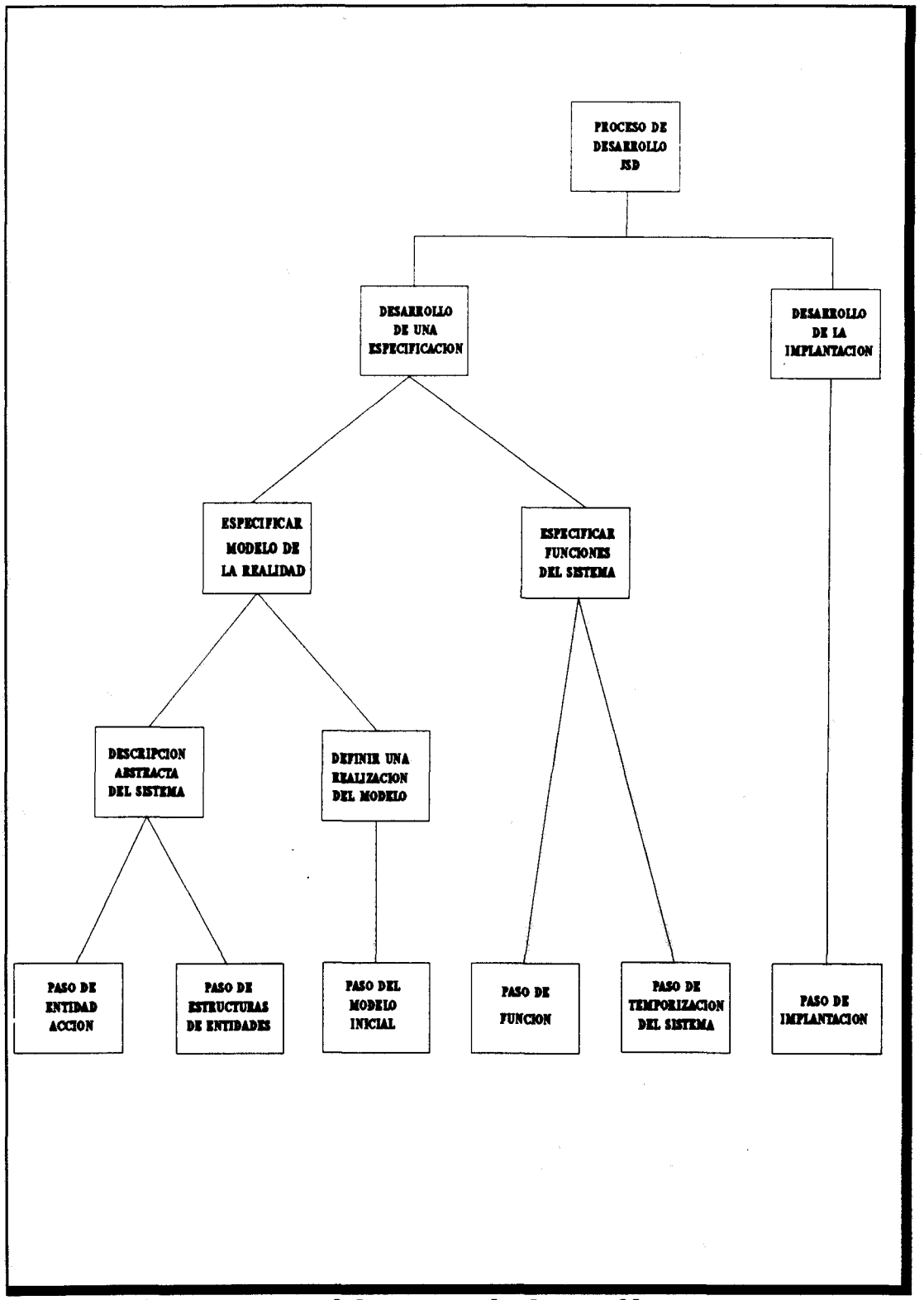


Figura 2.20 Etapas del proceso de desarrollo JSD

3.- Modelo inicial

El resultado de las dos etapas anteriores es la descripción abstracta del mundo real a la que se refiere Jackson. Es en esta etapa donde el diseñador comienza a especificar el sistema en sí, haciendo una simulación del mundo real. Esta se realiza estableciendo la forma en que se conectan los procesos en el mundo real (representados por las entidades) con los procesos del modelo, y trata los problemas que surjan, ya que la conexión es inevitablemente imperfecta. Esta conexión entre procesos (detallada en la técnica de diagramas de especificación del sistema) puede ser de dos tipos: por corriente de datos y por vector de estado. En esta etapa también se considera un subsistema de entrada para detectar, y si es posible reparar, los errores que eventualmente ocurren en la transmisión de los mensajes desde los procesos del mundo real hacia los procesos del modelo. La especificación de los procesos es realizado en texto estructurado.

4.- Función

Las funciones son añadidas al modelo para asegurar que se produzcan las salidas requeridas cuando ciertas condiciones de eventos ocurran. La adición de

funciones puede requerir cambios en el diagrama de especificación del sistema obtenido en la etapa anterior para crear nuevos procesos. En cualquier caso siempre tendremos que modificar el texto estructurado que ha sido elaborado para especificar las funciones requeridas.

5.- Temporización del sistema

En esta etapa se consideran las restricciones de tiempo. Estas pueden ser dadas por requisitos de usuario o bien por consideraciones técnicas. El resultado de esta etapa es una amplia especificación informal de las restricciones de tiempo que será muy útil para la etapa de implantación. Como consecuencia de este estudio, el diagrama de especificación del sistema contendrá unos procesos de sincronización cuya única función es la de asegurar que ciertas acciones han concluido satisfactoriamente antes de que se inicie otro proceso.

6.- Implantación

No entramos a detallar esta etapa por sobrepasar el alcance de la investigación.

Técnicas.-

Son tres las técnicas estructuradas empleadas en JSD:

- Diagramas de estructuras
- Diagramas de especificación del sistema
- Texto estructurado

DIAGRAMAS DE ESTRUCTURAS

El diagrama de estructuras permite expresar las tres estructuras clásicas: secuencia, selección e iteración. JSD emplea esta técnica para ordenar las acciones de una entidad en el tiempo. El rectángulo que hace de raíz del árbol representa a la entidad, y los rectángulos que desciende de él van a representar la estructura, en cuanto a su orden, de las acciones de esta entidad. Las hojas de este árbol son las acciones. Cualquier rectángulo en el diagrama que no sea una acción debe tener a su vez estructura de árbol.

TEXTO ESTRUCTURADO

Es esencialmente una transcripción de los diagramas de estructuras en forma de texto. Sobre esta base posteriormente se podrán añadir algunos elementos, tales

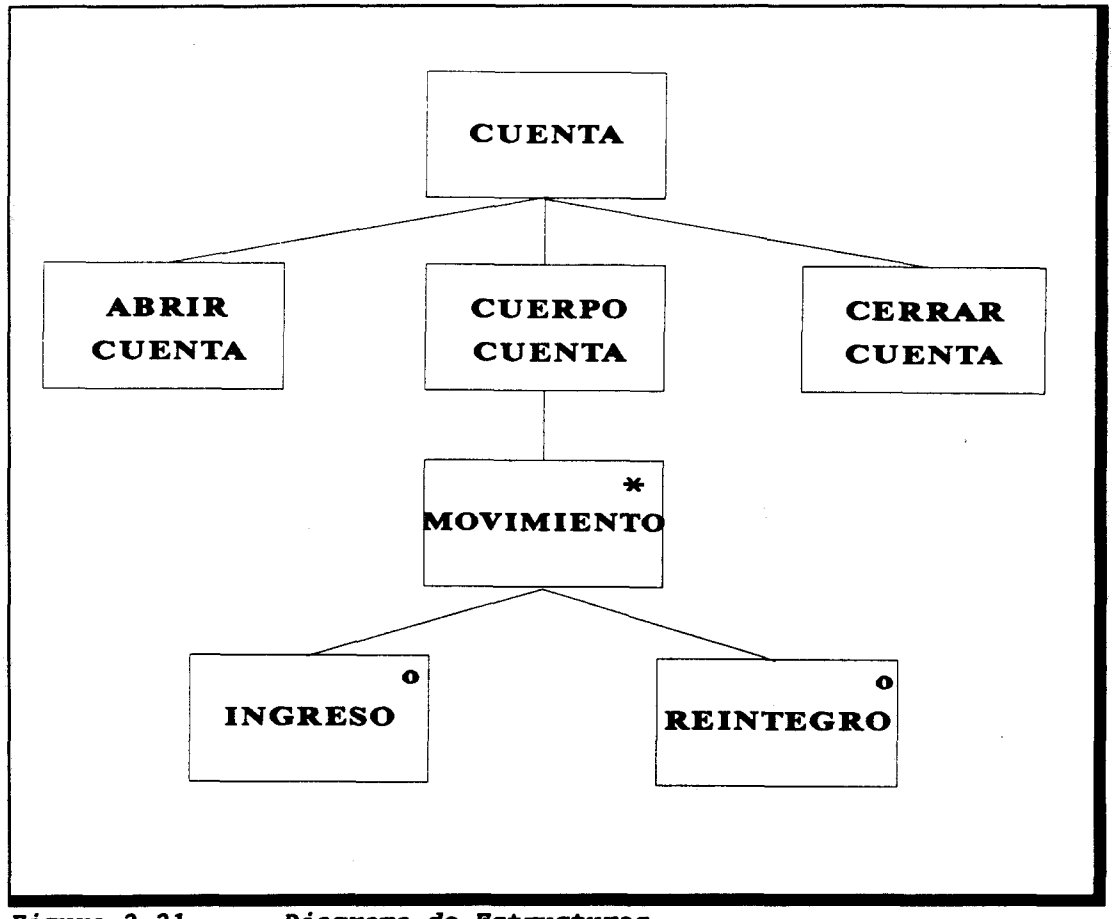


Figura 2.21 ***Diagrama de Estructuras***

como operaciones ejecutables y condiciones.

Ejemplo:

Secuencia	**Repetición**	**Selección**
X seq	**Y itr**	**Z sel**
...	**...**	**...**
...	**...**	**Z alt**
...	**...**	**...**
X end	**Y end**	**Z end**

DIAGRAMAS DE ESPECIFICACION DEL SISTEMA

Se utilizan en la fase del modelo inicial y sirven para mostrar la forma de conexión de los procesos del modelo a los del mundo real. Esta conexión puede ser por corriente de datos (un proceso escribe una corriente secuencial de datos y el otro proceso que se conecta lee esta corriente de datos) tal como se expresa en la figura 2.22. Los rectángulos representan a los procesos secuenciales, el círculo representa la corriente de datos y la flecha indica el sentido en que fluye la información.

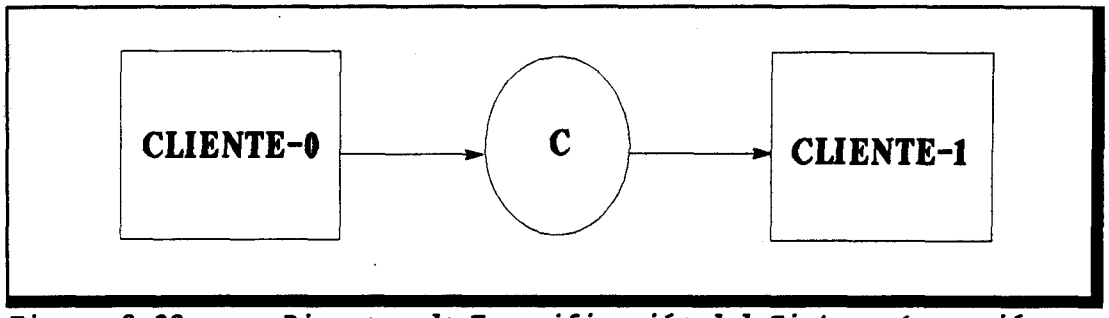


Figura 2.22 ***Diagrama de Especificación del Sistema (conexión por corriente de datos)***

La conexión por vector de estado (un proceso inspecciona directamente el conjunto de variables internas "vector de estado" del otro proceso) se representa como en la figura 2.23. Los rectángulos son procesos, el rombo indica que la conexión es por inspección del vector de

estado, y la flecha indica que el proceso al que apunta es el que inspecciona el vector del otro.

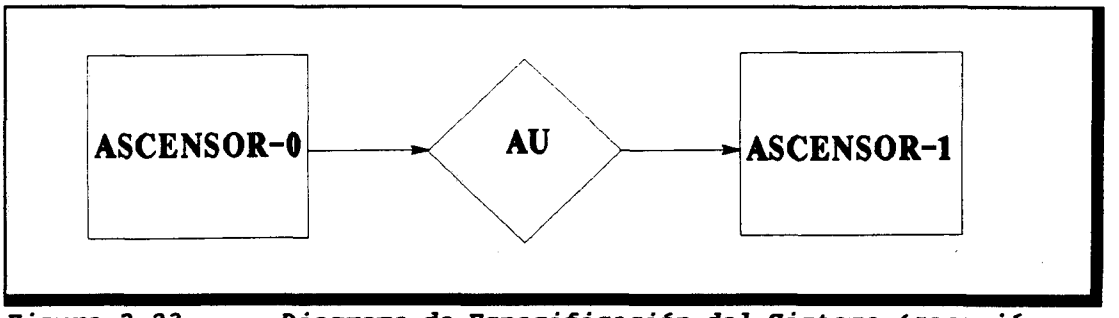


Figura 2.23 ***Diagrama de Especificación del Sistema (conexión por vector de estado)***

La multiplicidad en la conexión se indica mediante una doble barra en el lado de la flecha donde existen múltiples procesos en la conexión, ver figura 2.24. Aquí se expresa que la conexión es de uno a muchos, es decir un proceso CLIENTE está conectado a varios procesos CUENTA. También puede existir multiplicidad en la conexión de muchos a muchos.

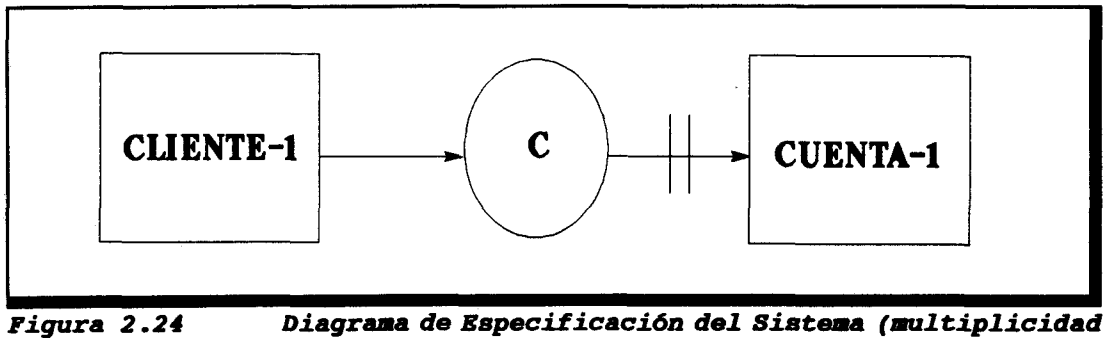


Figura 2.24 ***Diagrama de Especificación del Sistema (multiplicidad en la conexión)***

2.3.7.- INGENIERIA DE LA INFORMACION (IE)

Introducción.-

Los orígenes de esta metodología difieren según la fuente a la que se refiera. Parece ser que Clive Finkelstein fue el primero que usó este término para describir una metodología basada en el modelo de datos que desarrolló en Australia a finales de los setenta. En 1981 escribió una serie de artículos sobre la metodología. En ese mismo año colaboró con James Martin en un libro [MAR81]. A partir de entonces aparecen diferentes versiones de IE, las cuales son muy similares en el contenido pero con algunas pequeñas líneas diferentes. La razón de esto es que James Martin fundó una serie de compañías independientes basadas en la metodología de IE.

Fases y etapas.-

La principal base filosófica de IE es la creencia de que los datos son el corazón de un sistema de información y que además éstos son mucho más estables que los procesos que actuan sobre los datos. IE también considera los procesos en detalle, pero siendo los datos la base del sistema de información.

La metodología está compuesta de siete etapas divididas en cuatro fases, ver figura 3.5. Cada etapa tiene su propio objetivo:

- Planificación:

El objetivo es el de construir una arquitectura de información y una estrategia que soporte los objetivos de la organización.

- Análisis:

El objetivo es el de entender las áreas de negocio y determinar los requisitos del sistema.

- Diseño:

El objetivo aquí es el de establecer el comportamiento de los sistemas de la manera que el usuario quiera y que se pueda llevar a cabo con la tecnología existente.

- Construcción:

El objetivo en esta fase es construir los sistemas tal como han sido especificados en las fases previas.

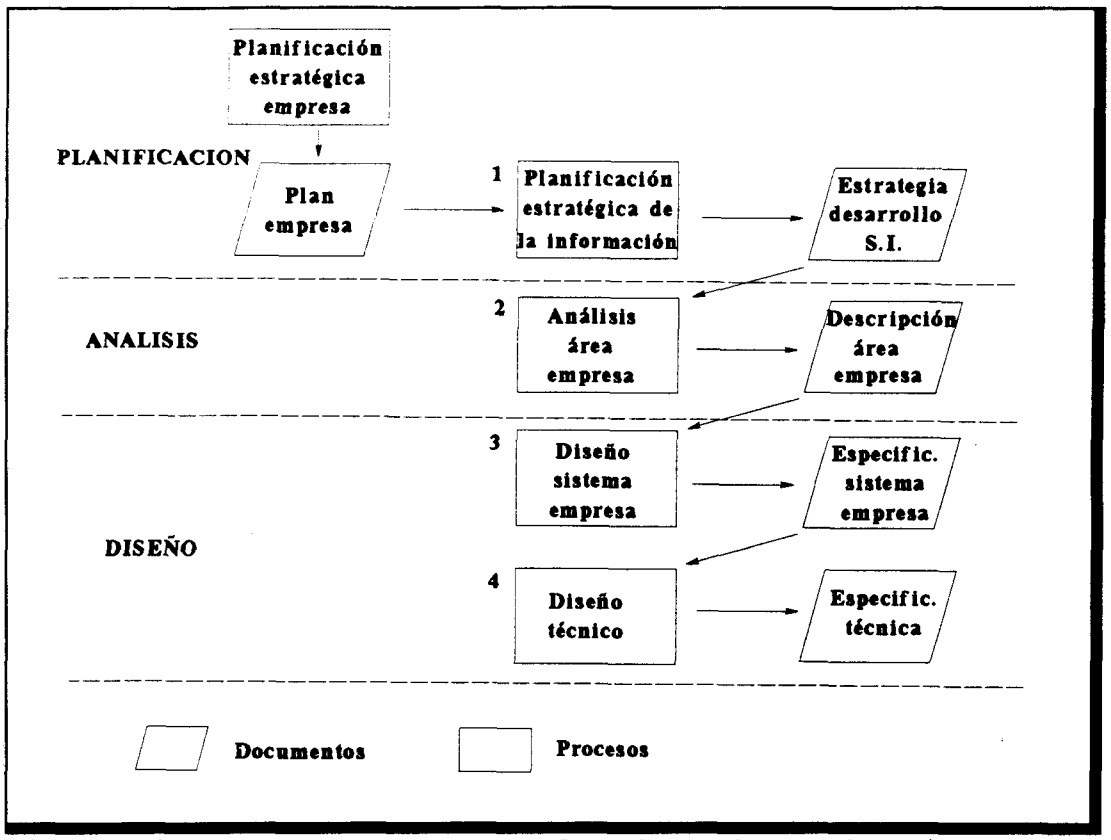


Figura 2.25 Fases y procesos de Information Engineering

Las etapas de Information Engineering son las siguientes:

1.- Planificación estratégica de la información

Se realiza un análisis general de los objetivos de la empresa, de la organización, de las funciones de principales de la empresa y de las necesidades de información. El resultado de este análisis es lo que se

llama "arquitecturas de información", que va a servir de base para los subsiguientes desarrollos y asegurar la consistencia y coherencia entre distintos sistemas en la organización. El plan de estrategia de información resultante documenta los requisitos de la empresa y las prioridades para el desarrollo de sistemas de información. Este plan hace posible que estos requisitos de alto nivel se tengan en cuenta a lo largo del desarrollo del proyecto. La realización de este plan implica cuatro tareas:

Análisis de la situación actual

Se estudia la organización y su posición actual, incluyendo los puntos fuertes y débiles de los sistemas actuales. Este estudio incluirá un análisis de la estrategia de la empresa, un análisis de la organización del sistema de información, un análisis del entorno técnico, y una definición de la arquitectura de información primaria.

Análisis de requisitos ejecutivos

Aquí los gestores enuncian sus objetivos, necesidades de información, prioridades, responsabilidades y problemas.

Definición de la arquitectura

Se identifica los tipos de entidades globales y la descomposición de funciones para cada área descrita en la etapa del análisis de la situación actual. Se realiza un análisis de distribución (los requisitos geográficos para las funciones y los datos), una identificación de los sistemas ideales requeridos en la organización, una dirección tecnológica requerida para soportar los sistemas y una organización de las funciones de los sistemas de información para soportar la estrategia.

Plan de estrategia de información

La creación de este plan incluye la determinación de las áreas de negocio (dividiendo las arquitecturas en grupos de negocio lógicos, cada uno de los cuales puede formar un análisis del proyecto), evaluación de la estrategia para llevar a cabo las arquitecturas, y un plan de estrategia para desarrollar los programas para los proyectos con más alta prioridad.

2.- Análisis del área

Las áreas de negocio identificadas en el plan de estrategia de información se tratan ahora individualmente y se realiza un análisis detallado de los datos y funciones. Las etapas de esta fase son:

Análisis de función y entidad

Se analizan los tipos de entidades y relaciones, así como los procesos y sus dependencias. En esta etapa se construyen diagramas de jerarquía de funciones y diagramas de dependencia de procesos.

Análisis de interacción

Examina la relación e interacción entre los datos y las funciones, y un análisis de la lógica de los procesos, y la preparación de los diagramas de acción de los procesos.

Análisis de la situación actual

En esta etapa se realiza un diagrama de flujo de datos y un modelo de entidades.

Confirmación

Es una verificación cruzada de los resultados

obtenidos en etapas anteriores.

Planificación para diseño

Identifica aquellas partes del modelo que van a ser automatizadas.

3.- Diseño de sistemas

Para cada área de diseño identificada en la etapa previa se diseña un sistema que recoja todos los requisitos de negocio identificados. En esta fase no se tienen en cuenta los factores técnicos, sólo se realiza un diseño lógico. En esta fase se realizan las siguientes etapas:

- Diseño de la estructura de datos preliminar
- Diseño de la estructura del sistema
- Diseño de procedimientos
- Confirmación
- Planificación para el diseño técnico

4.- Diseño técnico

En esta fase los aspectos de mecanización de los sistemas de información identificados en las fases previas se diseñan a un nivel técnico. Las etapas son:

- Diseño de datos
- Diseño software
- Diseño de transición
- Diseño de operaciones
- Verificación del diseño
- Diseño de prueba del sistema
- Plan de implantación

La fase de construcción no se describe por sobrepasar el alcance de la investigación.

Técnicas.-

Las técnicas empleadas en Ingeniería de Información son:

- Diagrama de descomposición
- Diagrama de entidades
- Diagrama de flujos de datos
- Diagramas de acción

Los diagramas de flujos de datos y los diagramas de entidad son idénticos a los empleados en la metodología SSADM.

DIAGRAMAS DE DESCOMPOSICION

Los diagramas de descomposición son apropiados para mostrar la estructura de una organización completa. El diagrama tiene estructura de árbol, en el que cada componente está representado por un rectángulo. Cada componente se subdivide para mostrar sus áreas funcionales.

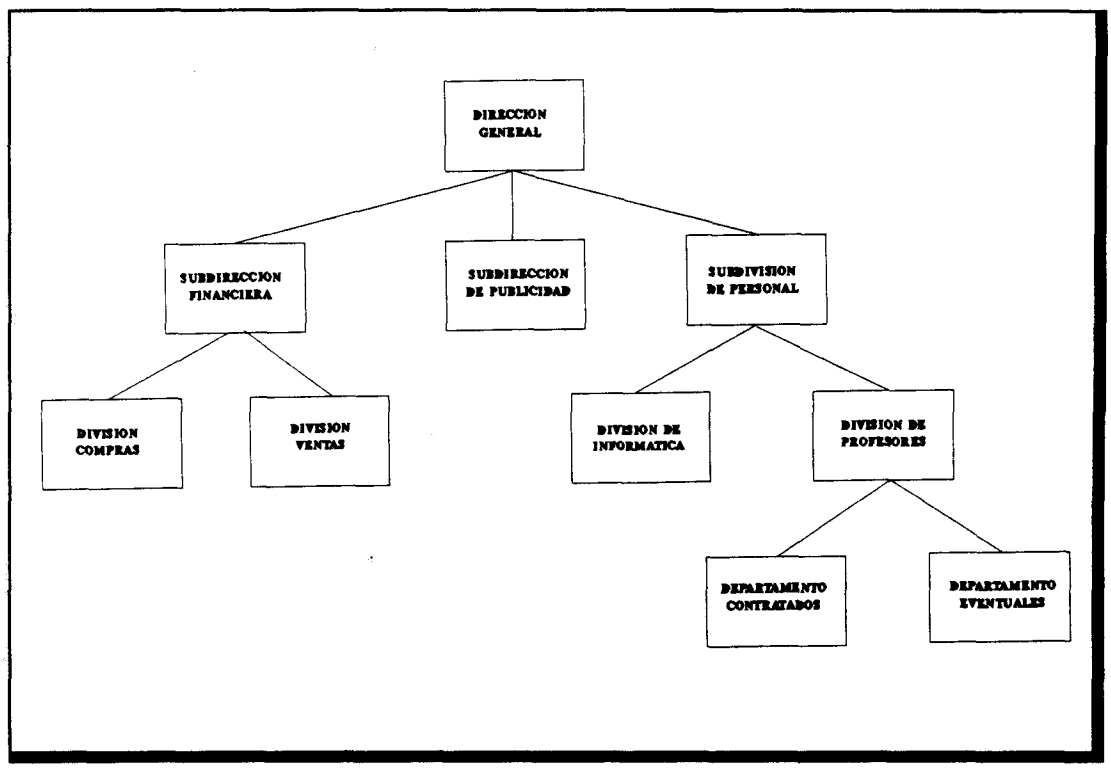


Figura 2.26 Diagrama de Descomposición

Las áreas funcionales se refieren a las principales áreas de actividad de la empresa (producción, distribución, etc.). Estas áreas funcionales se pueden subdividir en

procesos. Los procesos se refieren a grupos de actividades que son necesarias para el funcionamiento de la empresa ver, *figura 2.26.*

DIAGRAMAS DE ACCION

Los diagramas de acción sirven para especificar la lógica detallada de un proceso. Los corchetes son los

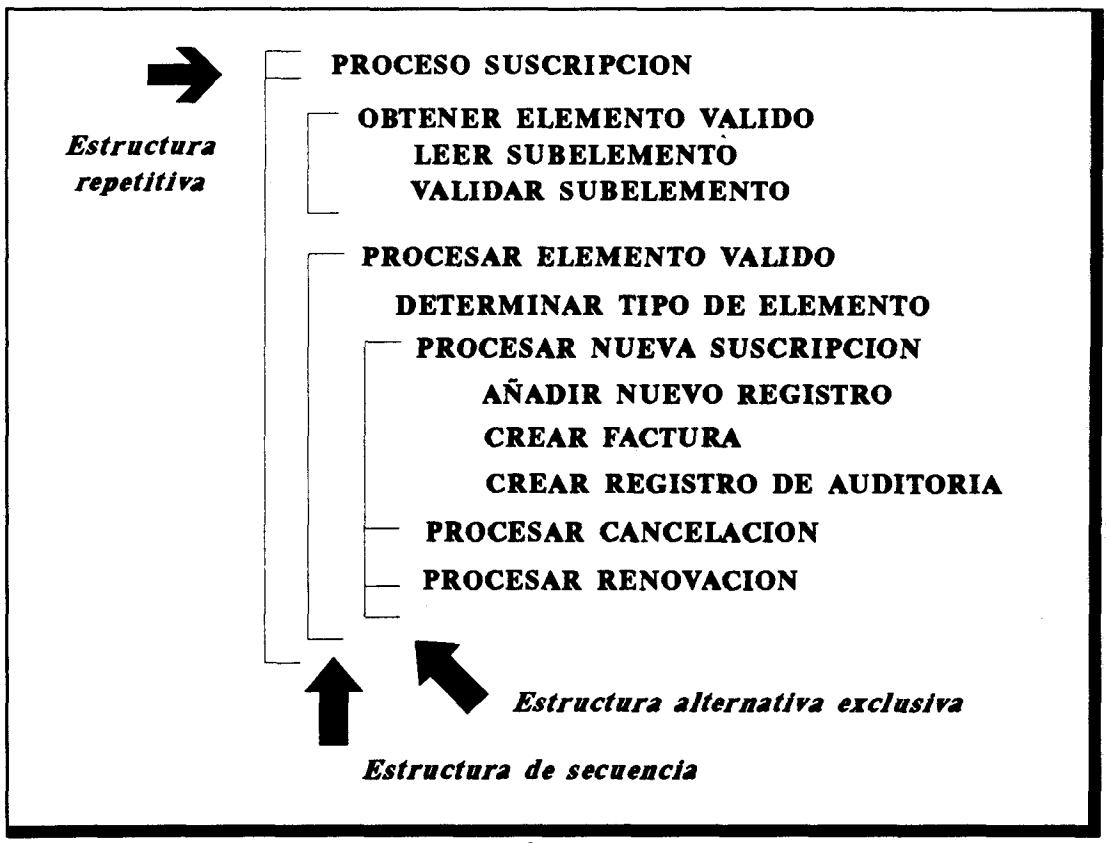


Figura 2.27 ***Diagrama de Acción***

bloques de construcción básicos. El corchete puede ser de cualquier longitud, de tal manera que se puede especificar tan detallado como se quiera. Dentro del corchete hay una secuencia de operaciones. Se entra por la parte de arriba del corchete, se hacen las cosas en una secuencia de arriba hacia abajo, y se sale por la parte de abajo del corchete. Puede haber dentro del corchete otros corchetes y pueden estar también anidados. La anidación muestra la estructura jerárquica de un programa. En la figura se muestra los tres tipos de corchetes que pueden aparecer en un diagrama de acción.

2.4.- CONCLUSIONES

En lo que respecta al uso de alguna técnica de análisis y diseño, el porcentaje de uso en los distintos paises es:

- Reino Unido (**31%**)
- Francia (**28%**)
- Alemania (**16%**)
- EEUU (**17%**)

Esto viene a demostrar que las diferentes técnicas de análisis y diseño son todavía ampliamente desconocidas por las personas que se dedican a desarrollar productos software. De hecho se ha visto en el análisis de estos cuatro paises que cuatro de cada cinco equipos de desarrollo no emplean ninguna técnica de análisis y diseño.

En lo que respecta a la incidencia de las diferentes técnicas de análisis y diseño, las más fuertemente introducidas en los EEUU son las de Yourdon, DeMarco y Chen; en el Reino Unido las de SSADM y Jackson y en Francia Merise.

Por último por lo que se refieren a la penetración de la cultura de desarrollo, en particular los aspectos de productividad y metodología, al primero se le da más énfasis en los EEUU mientras que a la metodología se le da más importancia en Europa.

Ambos factores se combinan positivamente en el Reino Unido y Francia; Alemania tiene pocos innovadores en este campo, importando muchas técnicas de análisis. En los EEUU la cultura de desarrollo no ha favorecido el uso de metodologías y sí de técnicas de análisis [EVA89].

DESARROLLO DE LA METODOLOGIA

CAPITULO 3

3.- DESARROLLO DE LA METODOLOGIA

La metodología CASE propuesta en este trabajo se basa en tres partes fundamentales:

- **Principios**
- **Técnicas**
- **Procedimientos**

En la primera parte se establecen los principios básicos de la metodología. Estos constituyen la base del desarrollo de la metodología MIDES diseñada.

En la segunda parte se detallan las técnicas que se emplean en la metodología MIDES. Estas se han seleccionado en función de:

- la concordancia con los principios establecidos
- la posibilidad de su automatización
- una integración adecuada entre las mismas
- el alcance de la investigación

En la última parte se expone:

- el ciclo de vida propuesto
- los formalismos y procedimientos que se emplean para lograr los objetivos marcados

3.1.- PRINCIPIOS BASICOS

3.1.1.- PRINCIPIOS DERIVADOS DEL CONCEPTO DE MODULARIDAD

Introducción.-

La modularidad es un concepto que se emplea para la resolución de problemas complejos. Consiste en la división del problema en un conjunto de problemas independientes y más pequeños, y por tanto más fáciles de comprender y resolver. Se ha demostrado empíricamente que de un problema X de complejidad C(X) tal que:

$$C(X) = C(X1) + C(X2)$$

el esfuerzo para la resolución del problema satisface la fórmula

$$E(X) > E(X1) + E(X2)$$

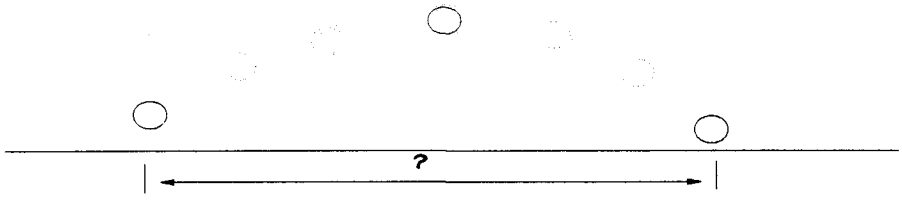
quedando en la realidad limitado este proceso por el esfuerzo de resolución de las interfases como se muestra en la figura 1.2, lo que da un intervalo óptimo de modularización.

Este concepto es un instrumento poderoso y esencial para el trabajo con problemas complejos. El motivo por el cual la modularidad empezó a aplicarse al desarrollo de sistemas fue precisamente porque la informática empezó a enfrentarse con problemas que por su envergadura, sobrepasaban las posibilidades de comprensión razonable por cualquier persona, lo que llevaba a elevadas cargas de desarrollo y mantenimiento.

"*La modularidad es un atributo simple del software que permite que un programa sea intelectualmente manejable*" [SHE81]. Permite, por tanto, que el encargado del desarrollo del software distribuya en componentes sencillos del sistema complejo, los cuales constituirán un todo integrado. Este concepto, aplicado a la construcción de programas, significa subdividir un programa en partes independientes, que sean de fácil manejo y a su vez puedan ser verificadas de forma independiente. Así pues, podemos decir que un programa es modular cuando cada parte del mismo puede ser desarrollada por un individuo en particular sin el conocimiento de las otras partes. De la misma forma, cuando el concepto se aplica al proceso de diseño de sistemas, significa dividir el proceso en pasos o etapas con objetivos definidos y de fácil control.

Un ejemplo de este concepto lo expuso Dijkstra [DIJ78], ver figura 3.1. Con este problema sencillo demostró que la modularidad, además de tener la ventaja de dividir el sistema y repartirlo entre un equipo de realizadores de software, tiene también como objetivo el dividir nuestras consideraciones y preocupaciones, de tal forma que se reduce el volumen total de razonamiento necesario y de hecho el volumen total del software.

Enunciado: Suponiendo un suelo plano y en el vacio (no hay resistencia del aire), si se tira una bala con una cierta velocidad inicial según un determinado ángulo, ¿a qué distancia caerá al suelo?.



Solución: *Problema horizontal:*

dist = Vx * t siendo Vx = velocidad horizontal
t = tiempo del trayecto

Problema vertical:

Vy = velocidad vertical con la que parte la bala
-Vy = velocidad vertical en el momento que vuelve a tocar el suelo

$t = \frac{2 * Vy}{g}$ siendo g la aceleración de la gravedad

Combinando las dos componentes se obtiene:

$$\mathbf{dist = \frac{2 * Vx * Vy}{g}}$$

Figura 3.1 ***Ejemplo de la aplicación* del concepto de Modularidad**

Desgraciadamente no existen unas reglas matemáticas bien definidas para la división de los módulos en el diseño, pero todos los tipos de organización en módulos comparten tres principios importantes: el principio de localización, el de la información oculta y el de abstracción [ROS75].

3.1.1.1.- PRINCIPIO DE LOCALIZACION

Todo código afectado por una decisión particular de diseño debe ser localizado dentro de un segmento específico del programa.

Extrapolándolo al diseño de sistemas, diremos que agruparemos todos aquellos elementos que tengan una relación lógica entre sí. Esto se aplica tanto a los datos como a los procesos.

Cuando este principio se sigue, implica que siempre que se modifique una decisión particular en el diseño y por tanto la correspondiente parte del sistema, esta modificación sólo afectará a dicha parte aislada dada la independencia de las partes en el diseño. Dicha modificación del segmento específico del sistema,

minimizará el coste de mantenimiento. En el ejemplo de la figura 3.1, Dijkstra señaló que si el Buen Dios nos hiciera la faena de modificar la ley de la caída de los cuerpos, modificando el valor de "g", todo lo que tendríamos que cambiar sería el problema vertical.

3.1.1.2.- PRINCIPIO DE LA INFORMACION OCULTA

Para tener control de la complejidad de un sistema, además de dividirlo en módulos, los enlaces que permiten la comunicación entre los módulos deben ser lo más simplificados posible.

El principio de la información oculta permite el cumplimiento de este objetivo, haciendo que cada módulo (parte del sistema) tenga conocimiento únicamente de la información que le es necesaria para cumplir su función. Esta información se hace inaccesible a otras partes del sistema. En un módulo sólo se visualiza la información necesaria para comunicarse con otros módulos. De esta manera, el principio facilita el mantenimiento del sistema, ya que se minimizan las partes del sistema afectadas por un cambio.

Pongamos un ejemplo: supongamos que se decide usar una estructura particular de una lista para una cierta tabla de datos, y esta decisión se asocia con un módulo del programa A. Si a un módulo B, externo a A, se le permite que haga referencia a la tabla usando conocimientos de la estructura, entonces B, podrá hacer cualquier cambio de la estructura, violando además de esta manera el principio de localización. Otro ejemplo, ver figura 3.2: el módulo B envia un mensaje requiriendo o proporcionando información para el módulo A, sin el conocimiento de su estructura interna, sólo sabe el protocolo de comunicación. Un componente del módulo A interpreta el mensaje y desencadena el funcionamiento de los componentes correspondientes que acceden a los datos y proporcionan los resultados a B.

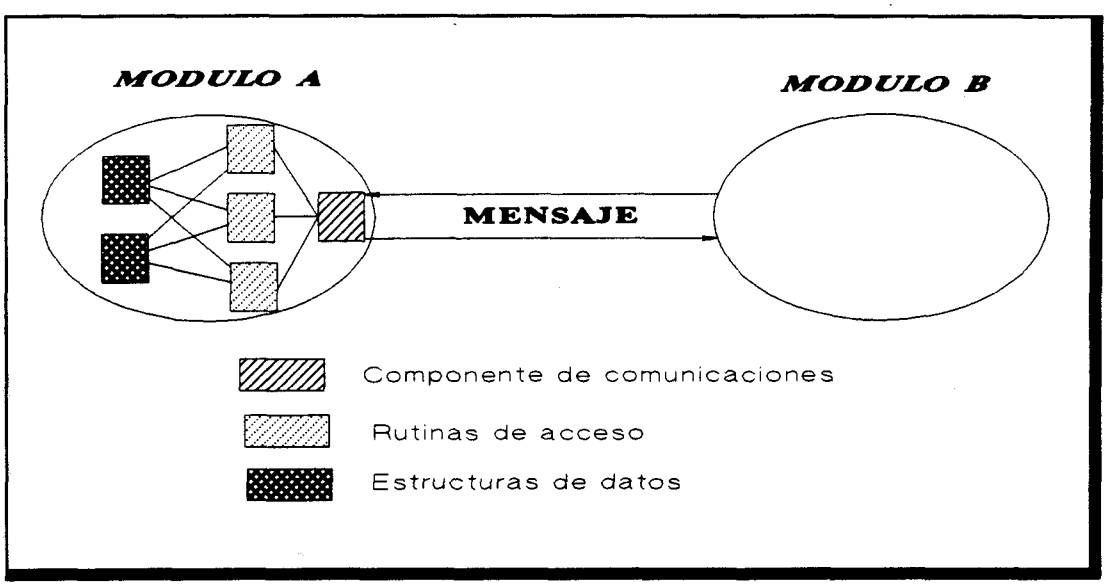


Figura 3.2 ***Ejemplo del Principio de Información Oculta***

3.1.1.3.- PRINCIPIO DE ABSTRACCION

Sin el principio de abstracción la revolución de las técnicas estructuradas nunca habrían podido llevarse a cabo. Abstracción es la capacidad de analizar un problema aislándolo de su entorno real, es una simplificación para poder describir ciertos hechos que se llevan a cabo, sin explicar cómo. Por medio de la abstracción nos podemos imaginar la solución de un problema sin considerar de inmediato las restricciones del medio, o aquellos detalles que lo afectan y son irrelevantes. Alguna de las mejores y más creativas soluciones a ciertos problemas se han obtenido de esta manera.

La abstracción es la mejor herramienta que se ha encontrado para trabajar con la complejidad que presentan los problemas que poseen muchas interrelaciones, ya que cada parte del sistema se define en un nivel dado de refinamiento, y se define solamente en términos de estas relaciones como una unidad para otras partes del sistema.

Un subsistema que se define así, se puede entender como una unidad, sin conocimiento de sus detalles y sin conocimiento de cómo este subsistema se emplea en niveles superiores. Las propiedades esenciales del subsistema se

clasifican, mientras que los detalles se omiten, de manera que son posibles diferentes alternativas de implantación.

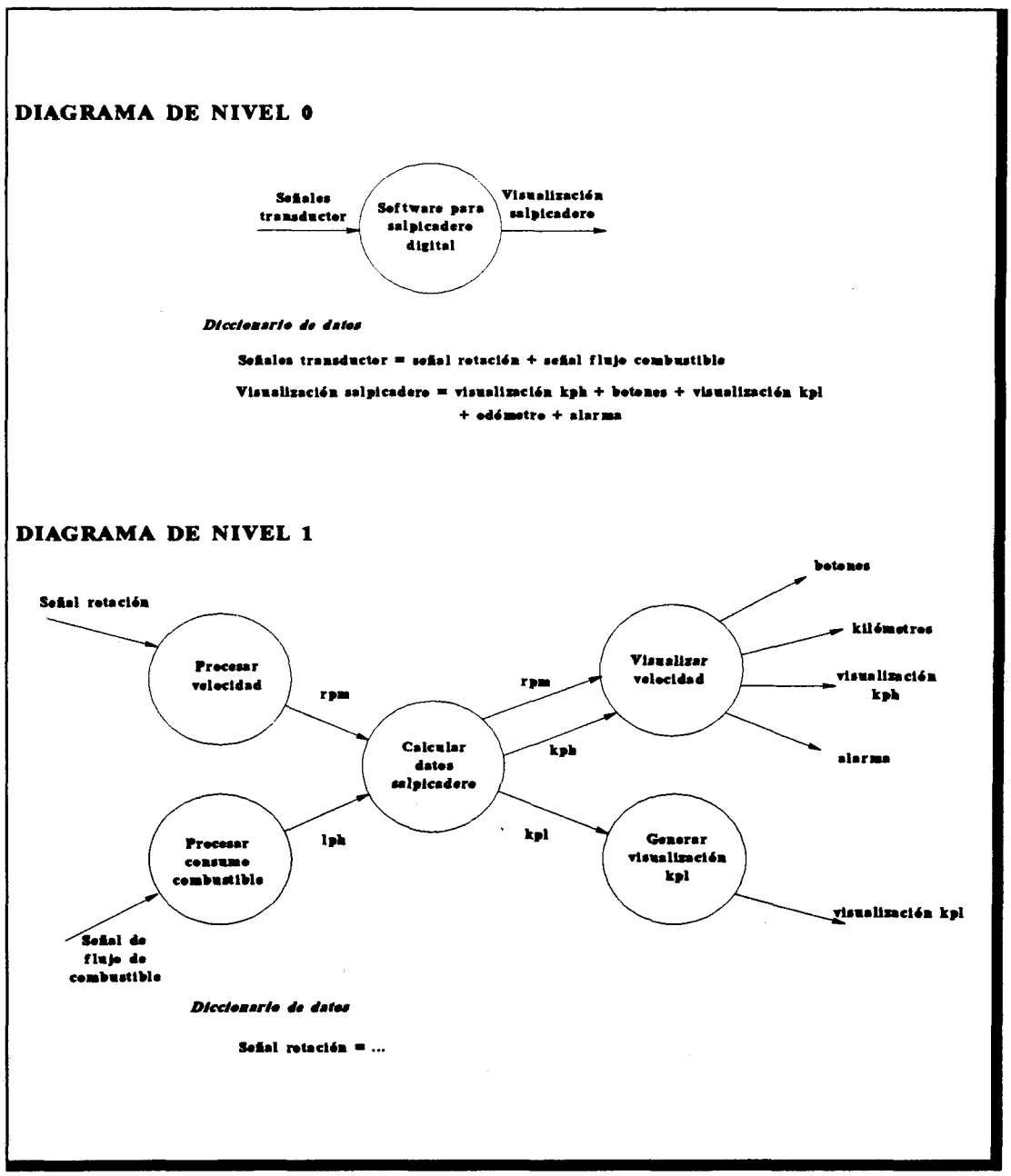


Figura 3.3 Ejemplo de aplicación del principio de Abstracción

Un refinamiento progresivo por niveles de abstracción me permite así solucionar los problemas complejos. Sólo en los niveles inferiores de abstracción se incorporan los detalles del problema.

3.1.2.- PRINCIPIOS DERIVADOS DE LAS TECNICAS ESTRUCTURADAS

Introducción.-

Para que los sistemas fueran desarrollados más rápidamente, a bajo coste y con calidad, y satisfaciendo así a una demanda creciente, los analistas y programadores debieron evolucionar desde métodos artesanales hasta disciplinas de desarrollo. Las técnicas estructuradas son el resultado de esta evolución y han contribuido en gran medida a una producción de calidad y mantenibilidad. Pero los problemas de productividad y calidad en el desarrollo de sistemas subsisten, ya que además de la demanda cada vez mayor de desarrollos, se hace necesario el manejo de sistemas cada vez más complejos. Por ello hoy se espera que estas técnicas sigan evolucionando hacia la automatización de modo que el desarrollo de las aplicaciones sea mucho más

rápido y eficaz. Ello implica el manejo no sólo de conceptos sino también de formalización de los mismos con el fin de que una herramienta los pueda manejar.

Los objetivos principales de las técnicas estructuradas son:

- obtener unos programas de alta calidad y comportamiento predecible
- obtener unos programas fáciles de modificar y de probar
- simplificar los programas y su proceso de desarrollo
- obtener una mayor seguridad de planificación y control en el proceso de desarrollo
- acelerar el desarrollo de los sistemas
- bajar los costos del desarrollo de sistemas

De un tiempo a esta parte estos objetivos se han visto ampliados, apareciendo los siguientes:

- obtener dentro de lo posible un diseño correcto, sin errores. Actualmente este objetivo está totalmente claro en los centros de investigación de informática. Para cumplir este objetivo, se estudia e investiga sobre nuevas herramientas que permitan desarrollar el análisis asistido por ordenador, con

lo cual se puede llegar a obtener diseños y programas que garanticen estar libres de errores, apareciendo de nuevo la formalización como requisito imprescindible.

- obtener una administración y análisis de los datos sólido. La administración y análisis de los datos no se consideraba en las primeras técnicas usadas, pero hoy se reconoce su vital importancia.

- dar al analista y programador unas herramientas de ayuda en el ordenador para desarrollar su trabajo.

- obtener el máximo de automatización en el diseño de sistemas con técnicas que hagan posible la generación automática de código. Los generadores de código eran escasos en los años 70, pero en estos momentos su uso se ha ido difundiendo, consiguiéndose una mayor velocidad de desarrollo, reducción de errores y facilidad de mantenimiento de los sistemas.

Filosofía.-

La aplicación de la filosofía estructurada comienza

con el mundo de la programación y desde allí se extendió al diseño y análisis de sistemas, para luego introducirse en áreas de planificación y administración de la información.

Esta filosofía comprende los principios relevantes para el manejo de la complejidad, ya sea en el diseño de software, diseño de circuitos integrados, o diseño de sistemas complejos; y también se refiere a la introducción de normas y disciplina, tanto para el proceso de diseño como para el producto final que se obtiene de este diseño. Para ello busca asegurar la introducción de procedimientos bien definidos, herramientas y técnicas de administración de proyectos, y mecanismos de comunicación en la administración del ciclo de desarrollo de sistemas, en el proceso de análisis y construcción de las aplicaciones, así como en el sistema resultante. Este conjunto de normas conforman el ciclo de desarrollo en una secuencia de pasos a seguir, revisar, controlar y documentar. Con esto se pretende obtener un orden que permita poder visualizar, en mejor forma el proceso de desarrollo de sistemas.

Podemos visualizar el paradigma de la producción de software en el siguiente gráfico

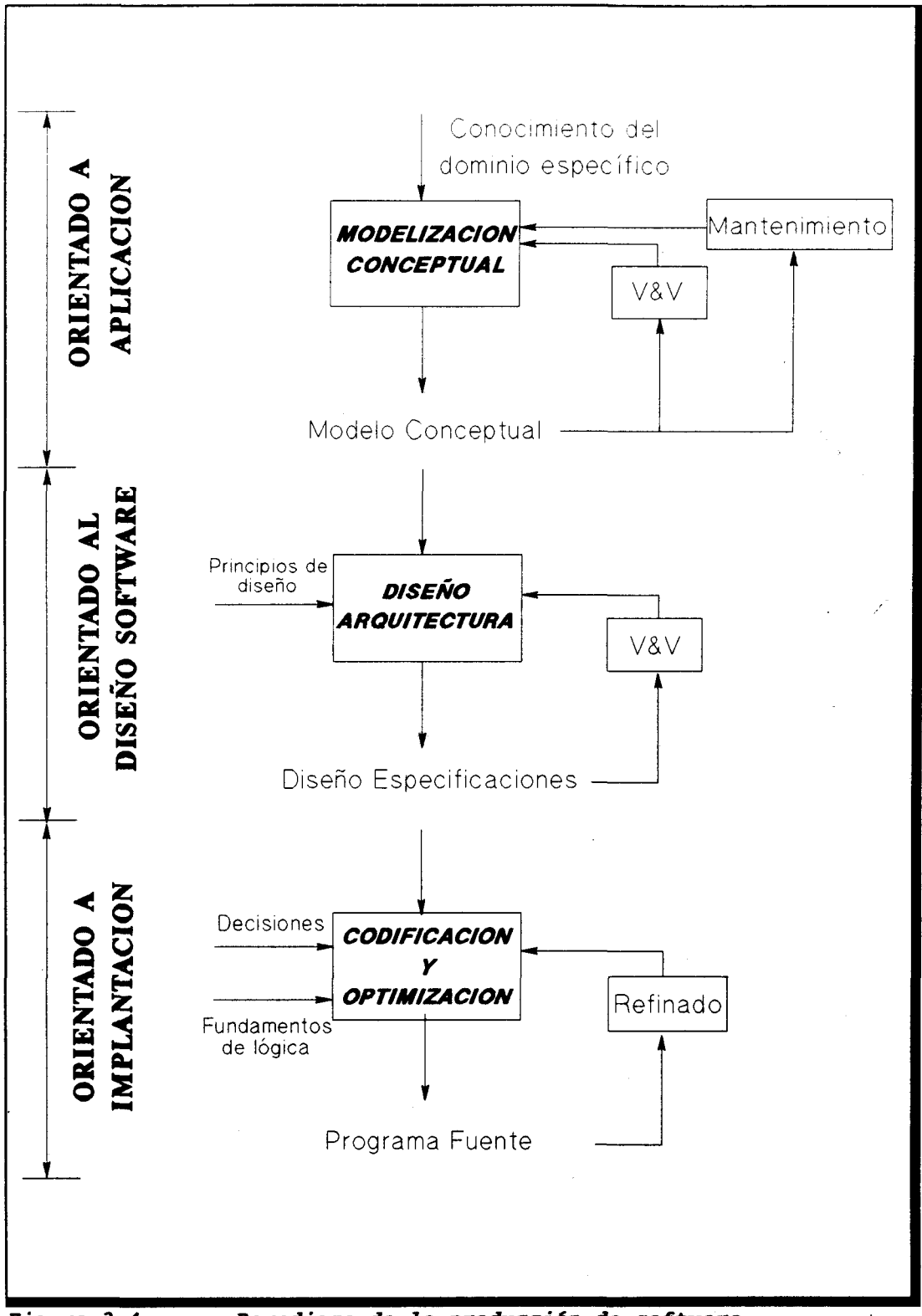


Figura 3.4 ***Paradigma de la producción de software***

La filosofía estructurada está compuesta por los siguientes principios para la resolución de problemas:

- principio de orden jerárquico
- principio de abstracción
- principio de formalidad
- principio de ortogonalidad

3.1.2.1.- PRINCIPIO DE ORDEN JERARQUICO

El principio de orden jerárquico está íntimamente relacionado con el de abstracción. Un sistema será más fácil de comprender, revisar y mantener, en la medida que esté dividido en partes o módulos. Pero esto no basta, se debe considerar además la forma en que están distribuidas cada una de estas partes. Esta distribución es de gran importancia para su comprensión. El organizar los subsistemas de un sistema en forma de árbol jerárquico aumentará la comprensión de éste. Esta forma de organización de las partes de un sistema permite obtener una construcción nivel a nivel, en que cada nuevo nivel agrega un mayor grado de detalle.

El principio de orden jerárquico se utiliza en muchos

sistemas, como por ejemplo en la organización de empresas. La organización jerárquica hace posible el manejo del control, la delegación y la comunicación de problemas, sobre todo en el caso de sistemas complejos que se componen de muchas partes. De la misma forma, la organización jerárquica puede ayudar a la solución y control de problemas complejos en grandes programas o sistemas. Otra ventaja que presenta el empleo de este principio es la de conseguir un mayor grado de minuciosidad en la prueba de los programas.

El principio de orden jerárquico, junto con el principio de abstracción que veíamos en el punto 3.1.1.3 es la base de muchas técnicas de análisis y diseño. Aunque es posible usar ambos principios en forma independiente, la combinación de ellos es el instrumento del análisis y diseño avanzado.

Combinando el principio de orden jerárquico con el principio de abstracción, podemos examinar un programa en forma de niveles o capas. El nivel más alto nos muestra lo más abstracto en una forma simplificada, mientras que el nivel más bajo nos presenta la atomización de los detalles de la realidad. Esto nos permite movernos poco a poco hacia los niveles más bajos, para ir analizando en forma gradual

los detalles de los componentes del sistema. Pero también nos podemos mover a través de los niveles en dirección ascendente, hasta que la solución del problema se pueda expresar en la forma de una poderosa instrucción de alto nivel.

Cada nivel de abstracción está compuesto por un conjunto de funciones agrupadas lógicamente. Las funciones que pertenecen a un mismo nivel, o a niveles diferentes, pueden comunicarse en base a dos reglas:

- los niveles más bajos no saben de la existencia de los niveles más altos, por lo tanto no tienen acceso a estos niveles. Sin embargo los niveles más altos pueden llamar a los niveles más bajos para ejecutar tareas.

- cada nivel tiene sus propios datos que pertenecen exclusivamente a las funciones de ese nivel, por lo tanto estos datos no pueden ser empleados por funciones de otro nivel. Los datos deben ser transferidos en forma explícita de un nivel a otro. Cada nivel puede tener acceso a una base de datos común, la cual debe ser estructurada formalmente.

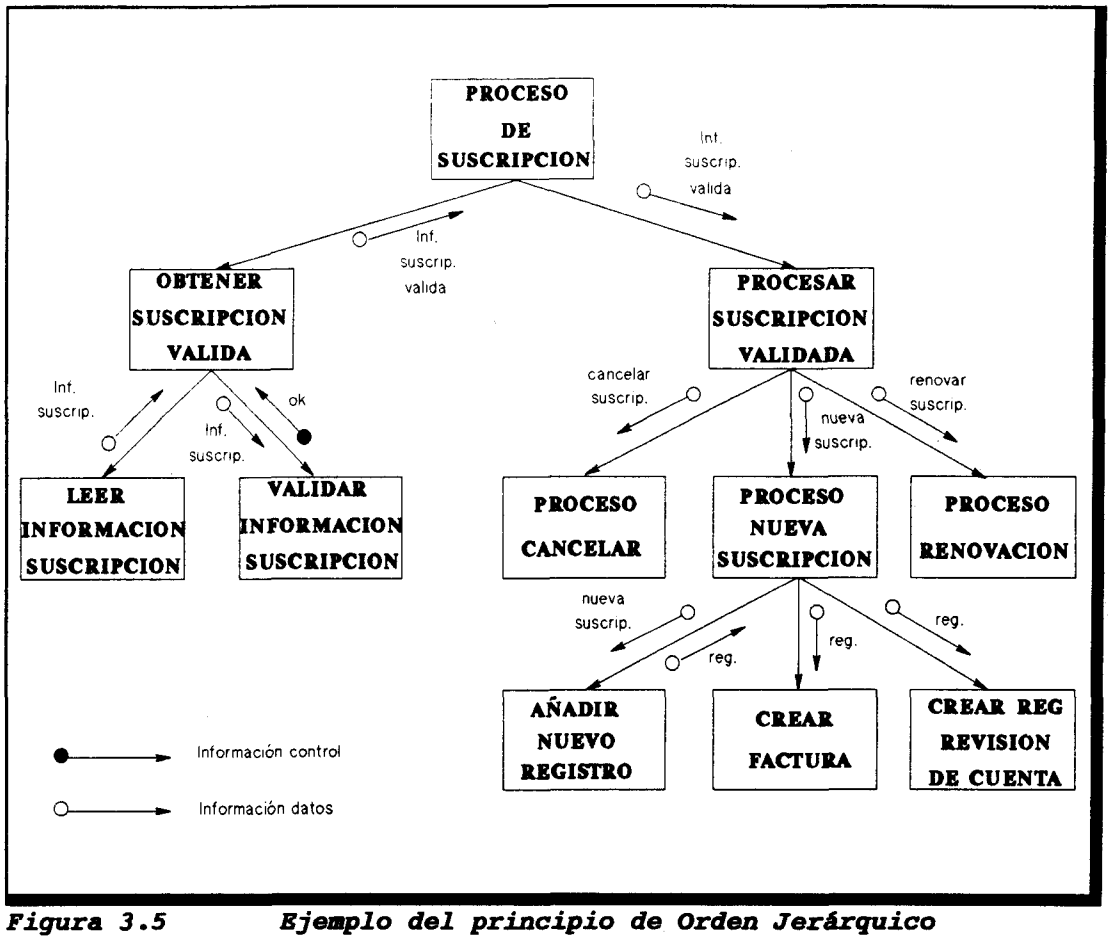


Figura 3.5 ***Ejemplo del principio de Orden Jerárquico***

Sin el uso del principio de orden jerárquico, el diseño estructurado sería poco más que el diseño modular.

Una de las diferencias esenciales entre el diseño estructurado y el no estructurado es que los diseños estructurados tienen un orden jerárquico, mientras que los no estructurados tienen sólo un orden secuencial. Una estructura jerárquica permite eliminar niveles superiores

sin afectar al uso de los niveles inferiores, cuyos módulos se pueden volver a utilizar para la modificación del sistema o para ser la base para un nuevo sistema. El uso de este principio puede ser un mecanismo poderoso para la creación de sistemas de fácil modificación.

3.1.2.2.- PRINCIPIO DE FORMALIDAD

La palabra formalidad nos sugiere una aproximación rigurosa y metodológica, lo cual es precisamente lo que se persigue en la filosofía estructurada. El principio de la formalidad es básico para conseguir la transformación del arte de programar en una disciplina similar a la ingeniería. Sin formalidad no existirían las bases para establecer los controles del proyecto, ni los propios controles de la calidad. Más aún, no existiría una base para determinar que un programa está correcto. La formalidad nos permite estudiar los programas (algoritmos), como objetos matemáticos. Finalmente, y tal vez lo más importante: la formalidad nos permite comunicar ideas e instrucciones de una forma exacta, es decir en una forma que se pueda automatizar.

La introducción de la formalidad en la programación

significa que los programas se guían por procedimientos sistemáticos, y que los resultados que se producen por cada procedimiento pueden ser rigurosamente definidos. Por otra parte, la formalidad considera un paso de evaluación, que permite determinar cuándo un procedimiento se ha ejecutado de forma correcta, y si el resultado que este produce es correcto. Esto no significa que la programación se reduzca a un proceso iterativo en el cual no intervienen la capacidad de resolver los problemas, ni la creatividad del ser humano.

La introducción de la formalidad en cualquier proceso, ya sea de orden técnico o de otro tipo, por lo general encuentra una cierta resistencia en aquellos niveles de fuerte desarrollo de la creatividad. Sin embargo la filosofía estructurada hace uso de la formalidad para dirigir la creatividad y encauzarla dentro del método y la disciplina. La formalidad es un camino positivo que tiene gran poder para combinar en la programación los aspectos creativos con los de la ingeniería en la programación con el objeto de producir software (producto de la mente) de análoga forma a la producción fabril en otros entornos de ingeniería.

El ciclo de diseño de aplicaciones es un ejemplo de

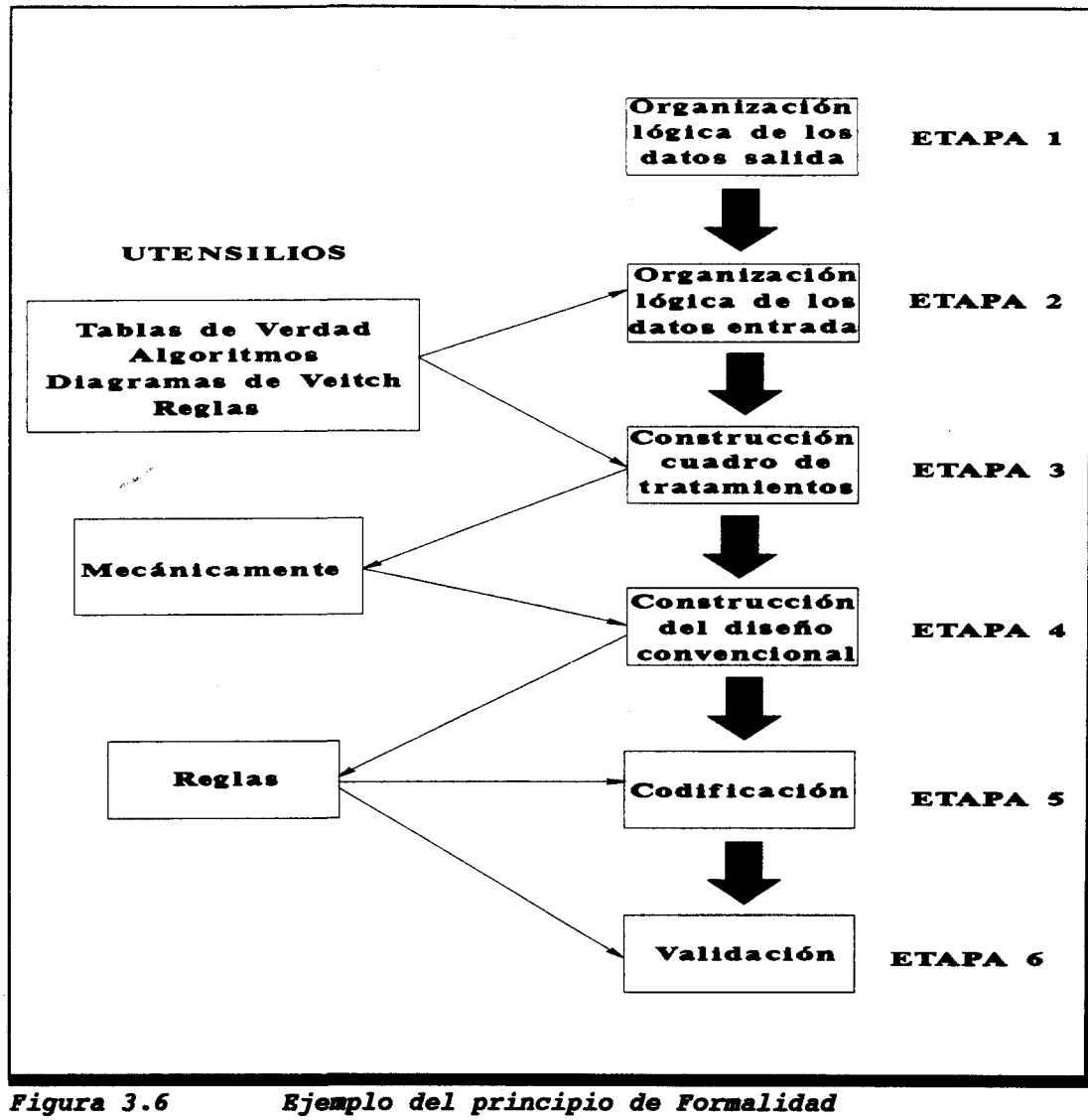


Figura 3.6 *Ejemplo del principio de Formalidad*

aplicación del principio de la formalidad, ya que define el proceso de desarrollo como una secuencia de fases. A su vez, para cada fase del ciclo se definen unos pasos en detalle. Existe una gran variedad de metodologías de diseño estructurado, cada una con sus propios procedimientos y pasos a seguir.

3.1.2.3.- PRINCIPIO DE ORTOGONALIDAD

El método de realizar un diseño o de escribir unas especificaciones de un proceso deberá ser altamente ortogonal [DEM79]. Es decir, que el conjunto de instrucciones a usarse para la especificación o el diseño de un proceso debería ser pequeño y simple. Además, nos debería obligar a expresarnos de una manera estandard.

En un artículo clásico de Bohm y Jacopini [BOH66] se demostra que un programa puede ser realizado con un lenguaje que utiliza únicamente estructuras de control. El concepto presentado en el artículo de Bohm y Jacopini referido como "Teorema de estructuras" es de fundamental importancia y sirve de base a la realización del diseño y análisis estructurado. Según estos mismos autores, son tres los bloques básicos de construcción necesarios para construir un programa:

1. Una caja de proceso
2. Un mecanismo de lazo
3. Un mecanismo de decisión binaria

La caja de proceso, mostrada en la figura 3.7a puede ser una sentencia simple de cualquier lenguaje, o una

secuencia de ordenes con una entrada y una salida.

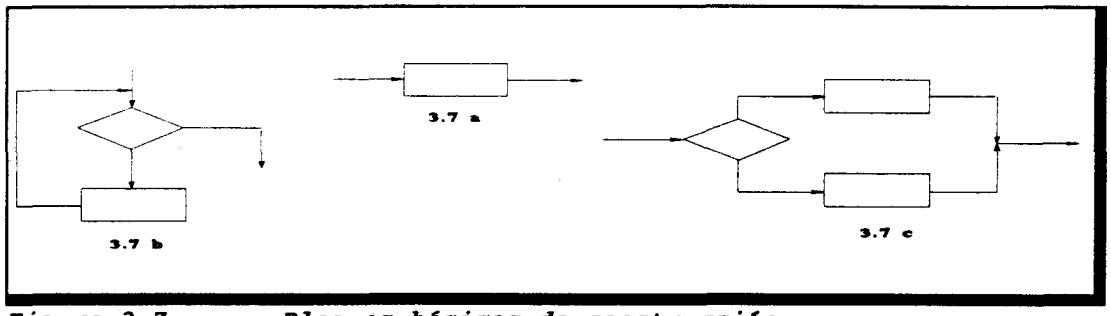


Figura 3.7 ***Bloques básicos de construcción***

El mecanismo de lazo, mostrado en la figura 3.7b, corresponde a lo que se conoce como "DO WHILE".

El mecanismo de la decisión binaria, mostrado en la figura 3.7c, corresponde obviamente a un "IF THEN ELSE".

Partiendo de estos tres bloques principales se pueden obtener los siguientes:

a) IF THEN: En este caso si la condición es falsa no se realiza ninguna instrucción, ver figura 3.8a.

b) CASE OF: Dependiendo del resultado de la variable se realizará una de las instrucciones siguientes, ver figura 3.8b.

c) DO UNTIL: Mientras la condición sea falsa se

ejecuta la instrucción, ver figura 3.8c.

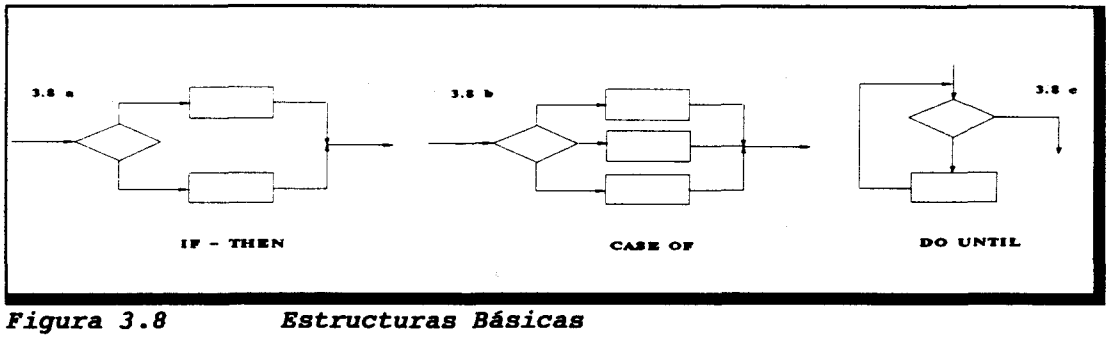


Figura 3.8 Estructuras Básicas

Aplicando el principio de abstracción conjuntamente con el principio de ortogonalidad, podemos observar que las construcciones mostradas en las figuras 3.7b y 3.7c pueden ser consideradas como una caja de proceso, ya que sólo tienen una entrada y una salida. Luego podemos definir una transformación de una operación de bucle a una caja de proceso, como se muestra en la figura 3.9a, y por tanto considerar cualquier operación de bucle equivalente a una caja de proceso (ligeramente más compleja). De una forma similar, podemos efectuar una transformación de una caja de decisión de la figura 3.7c a una caja de proceso, como se muestra en la figura 3.9b. Finalmente, podemos transformar cualquier secuencia lineal de cajas de procesos en una caja simple de procesos, como se muestra en la figura 3.9c.

Cualquier programa o proceso que esté compuesto de

cajas de procesos, operaciones de bucle, y construcciones IF THEN ELSE puede ser transformado sucesivamente, de acuerdo a las transformaciones mostradas en las figuras 3.9a, 3.9b y 3.9c a una única caja de proceso. Esta secuencia de transformaciones puede ser utilizada como una guía para el entendimiento del programa y para probar su exactitud.

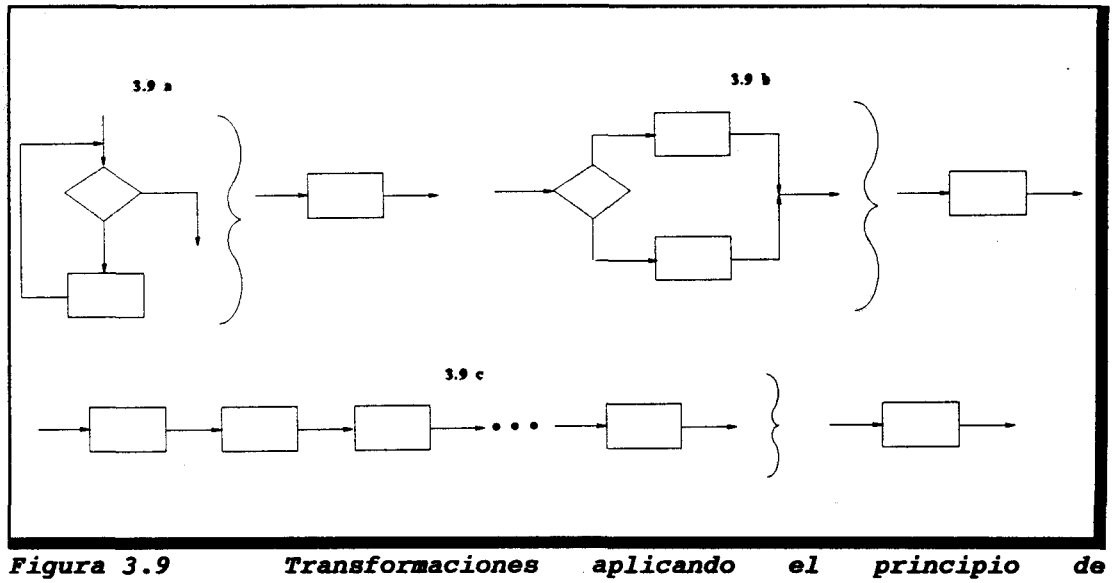


Figura 3.9 Transformaciones aplicando el principio de abstracción

El diseño top-down de un sistema se realiza aplicando la secuencia inversa de transformaciones, que acabamos de exponer, conjuntamente con el principio de orden jerárquico. Por ejemplo, comenzando con una caja de proceso simple, y expandiéndola gradualmente a una estructura compleja de estos componentes básicos.

Debemos notar la relación entre este concepto de "estructuras anidadas" y el concepto de modularidad discutido en el apartado 3.1.1. Un programa construido con las transformaciones que acabamos de mostrar es completamente modular. Cualquiera de las cajas de proceso puede ser reemplazada por otra, cuya función sea equivalente, sin afectar al resto del programa. Más importante es el hecho consistente en que las transformaciones de Bohm y Jacopini se pueden aplicar a un programa entero, descomponiéndolo en módulos más pequeños, los cuales pueden a su vez descomponerse en otros módulos más pequeños , y así sucesivamente. De acuerdo a este esquema, el proceso de transformaciones puede continuar hasta que hayamos alcanzado el nivel de módulos "atómicos": por ejemplo sentencias de programación, IF THEN ELSE o DO WHILE en cuyo momento la realidad tecnológica de solución al problema.

Esta es la diferencia esencial entre el diseño estructurado y el diseño modular. En el diseño modular el que realiza el software decide descomponer un sistema en módulos. Sin embargo, no intenta seguir descomponiendo cada uno de estos módulos en otros módulos más pequeños debido a la ausencia de reglas formales, quedando módulos que no pueden ser fácilmente subdivididos. Si los módulos no

pueden ser subdivididos, entonces deben ser tratados como una unidad, lo cual significa que cualquier prueba, mantenimiento, o intento de entender el módulo será complicado.

3.1.3.- PRINCIPIOS DERIVADOS DEL ENTORNO DE BASE DE DATOS

Introducción.-

A medida que fueron apareciendo los principios del diseño estructurado emergieron otras ramas de la informática: las bases de datos y los sistemas de información. Con la incorporación de estas nuevas técnicas quedó claro que la principal función del proceso de datos, orientado a aplicaciones de gestión, era obtener el dato correcto, para la persona indicada, y en el momento oportuno. Se presentó una imagen en que todos los empleados de una oficina poseían un terminal de ordenador sobre su escritorio, desde el cual podían consultar una base de datos. Así se difundieron los lenguajes de base de datos no procedurales y de fácil uso, los cuales no expresan cómo procesar los datos, sino más bien qué hacer con los datos.

Estos lenguajes, al menos para la mayoría de sus usos, no necesitan de las técnicas que se han derivado del diseño estructurado, sino que requieren que las bases de datos a las que ellos acceden estén bien diseñadas y bien implantadas.

Está claro que el funcionamiento de una empresa requiere de ciertos datos específicos, y que estos datos existen, independientes del diseño de cualquier proceso del sistema mecanizado que pueda usarlos. Los datos tienen una estructura inherente a ellos, la cual es independiente de cómo sean procesados. Un sistema de gestión de base de datos permite que los mismos datos sean utilizados por procesos diferentes, y que se puedan obtener informaciones diferentes a partir de los mismos datos, es decir los datos pertenecerán a varios dominios, constituyendo el núcleo de las interfaces. Ello exige una concepción global de los datos independiente de los tratamientos.

3.1.3.1.- PRINCIPIO DE LA INDEPENDENCIA DE LOS DATOS

Lo que acabamos de exponer conduce al principio de la independencia de los datos. Los modelos de datos representan:

- la estructura lógica inherente a los datos, que es descrita formalmente y en una forma independiente a las técnicas de acceso físico. Representan la visión global de la empresa.

- y la distribución de los datos que permiten las interfaces entre dominios.

En instalaciones de proceso de datos bien administradas, estos modelos de datos constituyen la base fundamental del proceso de datos. Una gran cantidad de programas diferentes utilizan subconjuntos de los mismos modelos de datos. Analistas y programadores usan un diccionario de datos común. Un administrador de datos, independiente de los proyectos de la corporación, es el responsable de la custodia de los datos y del diccionario. Esto hace necesario considerar el análisis de datos antes de diseñar los programas. Es ésta es una tarea básica, pero al mismo tiempo rigurosa, para un analista de sistemas. Los datos en cuestión deben ser analizados, y luego representados en los modelos de datos.

Por ello, es fundamental una planificación estratégica de los datos a través de toda la empresa, de modo que éstos, además de ser planificados, definidos y

estructurados, con el objeto de poder emplearlos en los diferentes procesos.

Los datos se convierten en un recurso estratégico de la empresa al igual que el dinero haciéndose necesario una administración de los mismos como ocurre con el dinero.

Se hacen necesarios lenguajes, herramientas y técnicas de administración, un tanto diferentes a las de los procesos de diseño tradicional, para permitir a los administradores y usuarios de una empresa obtener la información necesaria en los diferentes niveles en los que se puede presentar.

3.1.4.- PRINCIPIOS DERIVADOS DE LA INGENIERIA DE SOFTWARE

Introducción.-

La ingeniería de software [IEE83] "*es la aproximación sistemática al desarrollo, operación, mantenimiento, y retirada del software*".

Tanto la ingeniería de software, como las técnicas estructuradas, son una colección de métodos, técnicas y herramientas, y además comparten muchos de los objetivos y principios básicos. Sin embargo, la ingeniería de software es una disciplina mucho más amplia que las técnicas estructuradas; incluye a la metodología estructurada y a la colección de técnicas estructuradas, como también otras muchas disciplinas.

Destacamos siete principios básicos para asegurar el éxito de un proyecto de desarrollo software:

1.- Su dirección a través del uso de un plan de las fases del ciclo de vida de desarrollo.
2.- La realización de validaciones.
3.- El mantenimiento estricto del control del producto.
4.- El uso de técnicas modernas de diseño.
5.- El mantenimiento claro de responsabilidades de resultados.
6.- Emplear menos gente y mejor.
7.- Mantener un compromiso para mejorar el proceso.

Sólo podemos esperar el éxito de un proyecto software si se tienen en cuenta todos y cada uno de estos principios

que a continuación analizamos más detalladamente.

3.1.4.1.- LA DIRECCION A TRAVES DEL USO DE UN PLAN DE CICLO DE VIDA DE DESARROLLO POR FASES

Metzger [MET83] concluye tras el estudio de una serie de proyectos realizados sin éxito, que el cincuenta por ciento de los mismos ha fallado a causa de una pobre planificación.

Los puntos esenciales en un plan de proyecto software son:

- Vista general del proyecto:
  Es un sumario del proyecto que puede ser leido por cualquiera, por ejemplo un gestor de alto nivel, en unos pocos minutos y que indica los aspectos generales del proyecto.

- Plan de hitos de las fases:
  Aquí se discuten los productos a ser desarrollados en cada fase, sus actividades de desarrollo asociadas y sus planificaciones. Los productos principales de cada fase son planificados como "hitos" concretos, de tal

manera que no puede haber ambigüedad de si un "hito" se ha llevado a cabo o no, (ver figura 3.10).

- Plan de control del proyecto:

Incluye:

a) la organización del proyecto y las responsabilidades asociadas.
b) la descomposición del trabajo en tareas para controlarlas separadamente.
c) el plan de administración de recursos para controlar el gasto de todos los recursos críticos.
d) un plan de revisión de hitos y del proyecto que identifica las revisiones periódicas de los mismos, así como las tareas relacionadas que son esenciales en la revisión.

- Plan de control del producto:

Consiste en el seguimiento del producto a través del ciclo de vida software. Para ello identifica las principales actividades implicadas en el producto software, control de configuración y verificación y cómo estas actividades evolucionan a través del ciclo de vida del software.

- Plan de validación:

Expuesto en el punto siguiente.

- Plan de mantenimiento y operaciones:

Incluye una visión global de la operación del sistema y planes para entrenamientos, instalación, entrada de datos, facilidades de operación, obtención de salidas y mantenimiento de programas y bases de datos.

La importancia de este principio queda demostrada en los siguientes resultados: en un total de 151 auditorías realizadas sobre la adquisición y uso de sistemas de ordenador en los EEUU, las deficiencias encontradas se repartían del siguiente modo:

- 51% en la planificación de dirección
- 34% en el control
- 15% en factores tecnológicos

3.1.4.2.- REALIZACIONES DE VALIDACIONES

El principal propósito de este principio es detectar y corregir los errores cuanto antes.

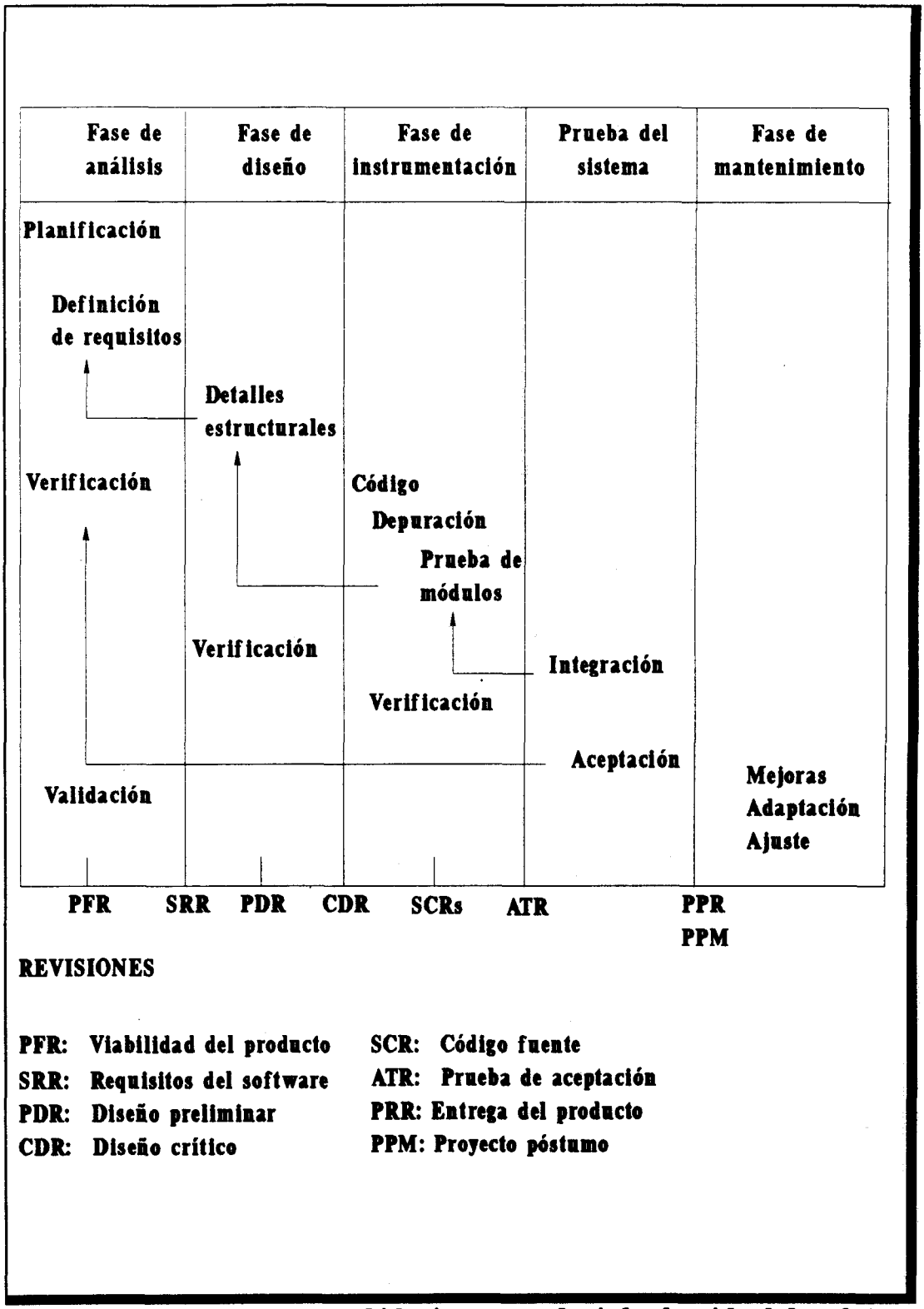


Figura 3.10 ***Fases y Validaciones en el Ciclo de Vida del Software***

Uno de los errores más frecuentes y costosos que se da en los proyectos software es el aplazamiento de la detección y corrección de problemas software hasta las etapas finales del proyecto, en la fase de prueba y validación final. Esto es un error por dos razones:

- los errores han sido hechos antes del comienzo de la codificación.

- cuanto más tarde sea detectado el error, más caro es corregirlo. Observemos la figura 3.11 [BOE76] donde se ilustra el costo relativo que supone un cambio en función de la fase en la que éste se haga.

Por tanto, cada fase del proceso de desarrollo de software deberá incluir una actividad de validación explícita, ver figura 3.10.

3.1.4.3.- MANTENIMIENTO ESTRICTO DEL CONTROL DEL PRODUCTO

Cualquier gran proyecto debe prever cambios en los requisitos a lo largo del ciclo de desarrollo. Esto es

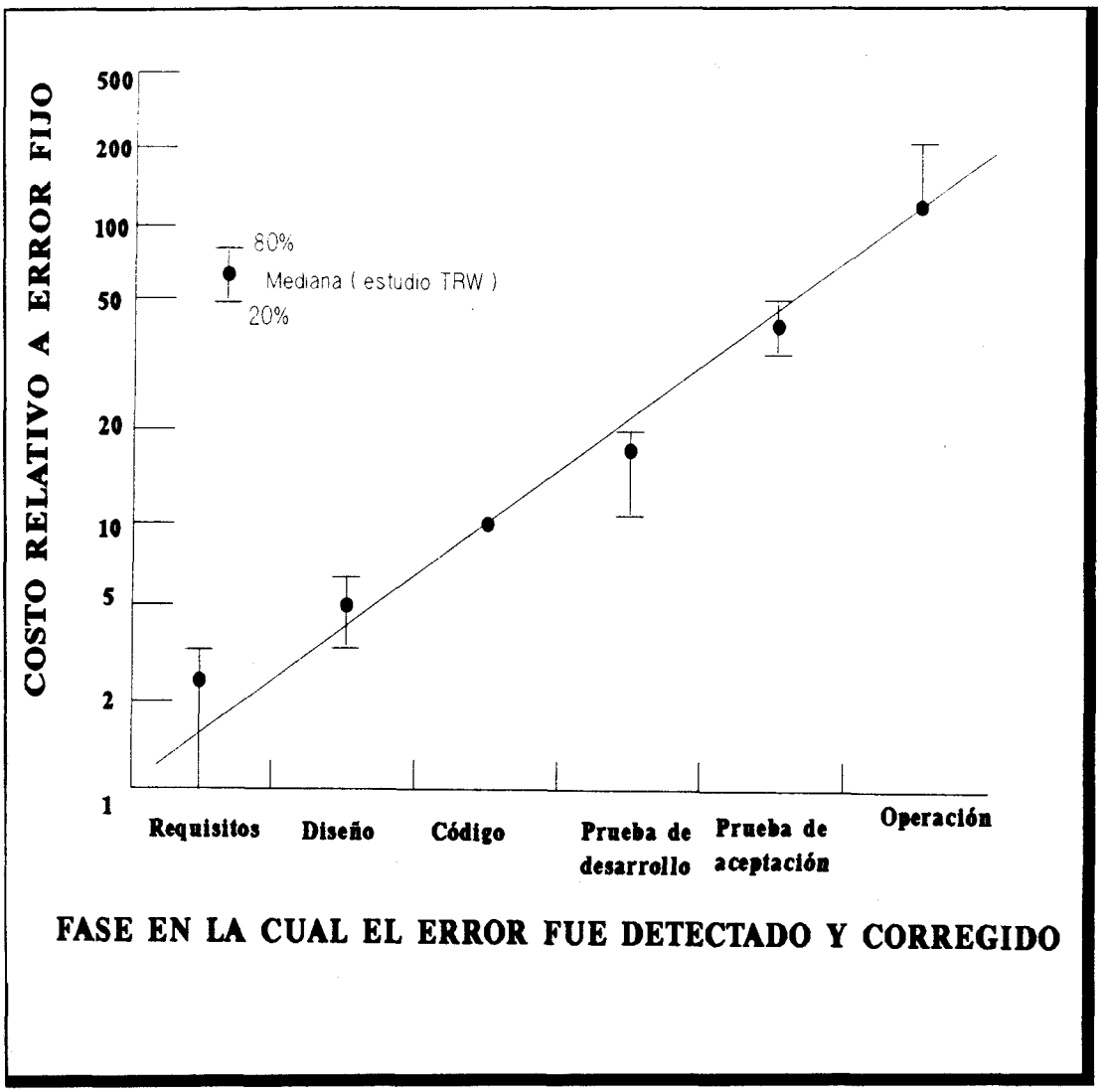


Figura 3.11 ***Costo relativo para realizar un cambio***

debido a que la comprensión del sistema a desarrollar es mejor a medida que se penetra en fases más detalladas. También puede deberse a cambios en el entorno externo, regulaciones de información del nuevo gobierno, mejoras en la tecnología, cambios de organización del usuario.

Desde el momento en que se incorporan estos cambios al desarrollo del sistema, la documentación y el código pueden multiplicarse.

El sistema más efectivo de control del producto software es la gestión de la configuración del documento base. El documento base es un documento o programa que experimenta una validación formal o proceso de aprobación y que sólo puede ser modificado posteriormente mediante procedimientos formales.

3.1.4.4.- EL USO DE TECNICAS MODERNAS DE DISEÑO

El uso de estas técnicas incluye la filosofía estructurada (apartado 3.1.2) y las de diseño orientado a objetos. Dichas filosofías contribuyen enormemente a la detección y corrección de errores al inicio, produce mucha más comprensibilidad, mejor mantenimiento del producto software, y hace que otros trabajos software, integración y prueba, sean más fáciles.

El uso de estas técnicas modernas de diseño no significa un sustituto para los otros principios básicos, los cuales serán componentes de estas filosofías.

3.1.4.5.- MANTENIMIENTO CLARO DE RESPONSABILIDADES DE RESULTADOS

Un proyecto que siga los cuatro primeros principios aún puede tener problemas. La figura 3.12 traza el tanto por ciento de trabajo efectuado estimado dado en informes de progreso semanal en un proyecto. A los directivos a los que se les den tales estimaciones pueden pensar que han enviado a sus programadores a un túnel profundo y oscuro.

Se debe dar a los miembros del equipo del proyecto un informe claro de los resultados de los que ellos y su grupo serán responsables. Así mismo han de saber que las recompensas futuras dependerán de aquellos resultados. Para mantener tal responsabilidad se requiere suficiente visibilidad en la ejecución del proyecto de cada persona o grupo. Para ello en el momento actual existen técnicas de planificación y control del desarrollo soportadas en herramientas CASE que proporcionan información real y fidedigna de la situación del proyecto.

Esta visibilidad es particularmente difícil de conseguir en los proyectos de software sin el empleo de tales productos. Para lograr esta tangibilidad y exactitud del producto software, el proceso de software debe ser

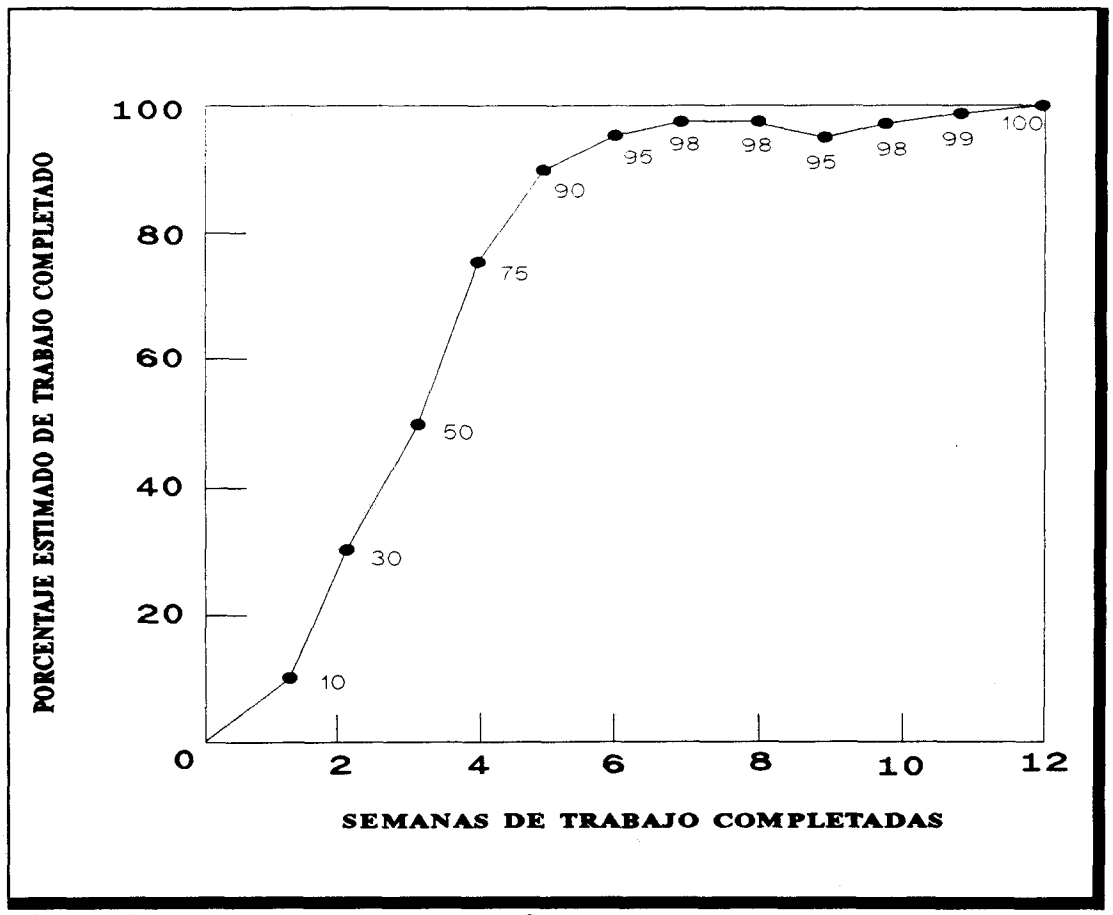


Figura 3.12 ***Progreso semanal en un proyecto***

dividido en un gran número de pasos bien definidos. El proyecto debe estar organizado en torno a estos pasos de modo que las herramientas proporcionen el control deseado.

Estos pasos deben consistir no sólo en componentes perfectamente delimitados, sino también en acuerdos formales entre subgrupos. Un ejemplo de estos acuerdos podría ser una autorización escrita del trabajo del

proyecto entre la organización del proyecto y la organización de la ejecución, identificando la distribución del subproyecto específico y sus planificaciones y presupuestos asociados.

3.1.4.6.- EMPLEAR MENOS GENTE Y MEJOR

Una razón para este principio es la ancha escala de productividad de desarrollo software entre los individuos. Se ha denunciado una variación de 26:1 en el tiempo requerido para completar una asignación de programa. Otro experimento encontró una variación entre individuos de 10:1 en el número de errores que quedaban en un programa acabado. Los análisis de la productividad de diseños estructurados han mostrado una variación típica de 5:1 entre individuos.

Es importante reducir la pérdida de productividad debido a las comunicaciones en un proyecto, consiguiendo que el trabajo se haga con menos ejecutores y más productivos. Un grupo de n miembros tiene un potencial de n(n-1)/2 caminos de comunicación. Una organización jerárquica de un proyecto reduce este número, pero el efecto es todavía considerable.

Además existen actividades que deben desarrollarse en serie y el hecho de incorporar más recursos no incide en el plazo. No hay que confundir meses con meses/hombre. "Un niño no lo pueden tener nueve mujeres en un mes" [BOE81].

Las técnicas de planificación modernas hacen un análisis de las actividades y establecen el número óptimo de recursos y cuándo deberán incorporarse. El separarse de estos valores supone más tiempo y mayor coste, como se ve en el gráfico [BOE85].

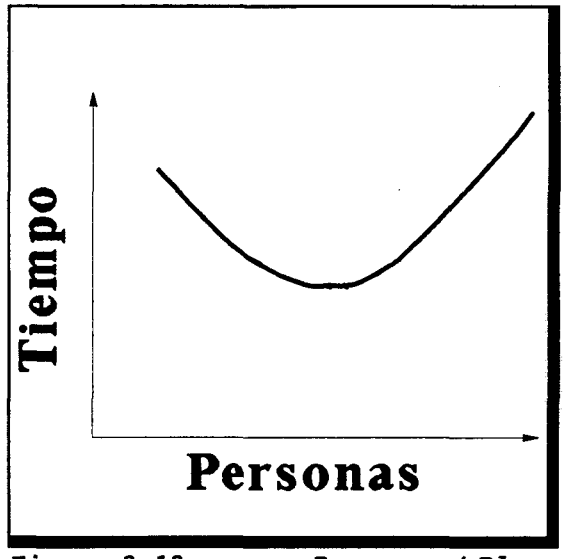


Figura 3.13 ***Recursos / Plazo de Tiempo***

Los directores no deben:

- intentar resolver los problemas de fechas del proyecto añadiendo más gente. Esto añade una sobrecarga de comunicación, particularmente porque las nuevas personas deben entender el proyecto.

- sobrecargar un proyecto con gente durante las etapas

iniciales. Este es el periodo durante el cual las comunicaciones son más intensas, y los requisitos y diseño son más volátiles. Demasiada gente implicada en este momento significa una pérdida de esfuerzo pues entre otras cosas hay que preocuparse de darles algo que no servirá para nada.

Los directores deben:

- establecer escalas de salarios u otros beneficios para recompensar a los individuos con un alto grado de productividad.

- apartar aquellos sujetos cuya productividad es baja, insertándoles en otras actividades de la empresa.

- emplear herramientas CASE en el proceso de desarrollo software. La automatización reduce el número de personas requeridas y la sobrecarga de comunicación. La automatización, además, ayudará a producir especificaciones y diseños con menos errores, más fáciles de probar, modificar y usar siempre que se utilicen las técnicas adecuadas, y siempre y cuando estas herramientas sean un soporte de la metodología elegida.

3.1.4.7.- MANTENER UN COMPROMISO PARA MEJORAR EL PROCESO

Un compromiso para mejorar el proceso no implica sólo un compromiso para probar nuevas técnicas software que parecen prometedoras, sino también para establecer un plan para evaluar el efecto de usar nuevas técnicas. Esto implica que el departamento tenga un método de recolección y análisis de datos sobre su eficiencia con y sin las nuevas técnicas, métodos o herramientas.

La colección de datos puede emplearse como una base para determinar dónde están los cuellos de botella en los proyectos, dónde va la mayor parte del dinero, y dónde son más pobres las estimaciones.

Estos datos pueden ser usados para determinar cómo funcionarán las cosas cuando se empleen nuevas técnicas y para estimar los costes de futuros proyectos de software.

3.1.5.- PRINCIPIOS DERIVADOS DEL DISEÑO ASISTIDO POR ORDENADOR

Introducción.-

En un comienzo, las técnicas estructuradas fueron concebidas para el diseño en forma manual. En la medida que el diseño de sistemas y la programación emplean métodos manuales, están restringidos a metodologías que puedan ser afrontadas por el hombre normal. Se han sugerido metodologías más poderosas, pero son difíciles y complejas de aplicar en forma manual, estas metodologías necesitan ser automatizadas.

El cerebro humano cuando maneja una gran cantidad de detalles está expuesto a cometer errores: errores de programación, errores al hacer un modelo de datos, etc.; lo que trae como consecuencia que las entradas y salidas no se den la mano en diagramas de flujo, errores, ambigüedades, omisiones e inconsistencias en las especificaciones. Se deben diseñar técnicas estructuradas que sean verificadas y corregidas por el ordenador, de tal modo que pueda controlar que las entradas y salidas se correspondan entre sí, pueda detectar las omisiones e inconsistencias, pueda generar automáticamente el modelo

los datos lógico y físico y el código de los programas.

Existen en estos momentos herramientas CASE que permiten realizar estas tareas, las cuales deben:

1) ser integradas en mejor forma
2) tener una mejor interacción con el usuario
3) estar pensadas para ser usadas por cualquier analista de sistemas.

3.1.5.1.- EVITAR LA PROGRAMACION CUANDO SEA POSIBLE

La programación es la actividad que requiere una mayor intensidad de trabajo, y está expuesta a que se cometan errores. Estos errores se pueden omitir usando generadores de programas. En estos momentos se usan generadores de informes, de gráficos, de diálogos por pantalla, de accesos a bases de datos, y en algunos casos se generan aplicaciones completas.

Lo deseable es que existan generadores para más tipos de aplicaciones y lenguajes de especificación en vez de lenguajes de programación. Los lenguajes de especificación permiten establecer qué es lo que se desea obtener, sin

tener que construir el programa en detalle. Las especificaciones creadas por medio de estos lenguajes deben ser lo suficientemente precisas como para que se pueda generar el código del programa a partir de ellas.

Por lo tanto deberá ser un principio del diseño asistido por ordenador el que la programación se evite siempre que sea posible. Esto no invalida los principios descritos anteriormente, por el contrario, a medida que diseñemos sistemas más complejos, estos principios irán adquiriendo mayor importancia, ya que se aplican a los procesos de diseño y especificación de los sistemas, tal como a la construcción de sistemas.

3.1.5.2.- EMPLEO DE HERRAMIENTAS GRAFICAS

El diseño asistido por ordenador necesita de buenas herramientas gráficas. El analista y el usuario final deben poder comunicarse a través de los diagramas de los sistemas, los cuales a su vez deben ser del mayor contenido y facilidad de comprensión posibles. Estos diagramas normalmente han sido diseñados sobre simples hojas de papel. La idea es que puedan ser creados, editados y manipulados en un monitor de ordenador, lo que permite una

facilidad de mantenimiento y análisis de consistencia. En este caso hay que buscar las técnicas de diagramación que sean más apropiadas para ello, es decir que soporten el principio de orden jerárquico, que hemos visto en el apartado 3.1.2.1. El diseño gráfico debe permitir la descomposición de los diagramas en diferentes grados de detalle, de modo que se aplique el principio de abstracción, visto en el apartado 3.1.1.3, hasta llegar al punto en que el código pueda ser generado directamente a partir del diagrama.

El diseño a través de gráficos se debe implantar de tal forma que pueda ser verificado de inmediato, cumpliendo así el principio de continuas validaciones. El diseñador debe poder recibir un aviso inmediato cuando comete un error. Las inconsistencias, omisiones y errores se deben resaltar en la pantalla. Además el diseñador debe poder obtener la ayuda necesaria para efectuar la corrección del error detectado.

3.1.5.3.- LOS LENGUAJES DEBEN SER CONSTRUIDOS PARA LAS TECNICAS DE DISEÑO

A comienzos de los años 70 se inició una corriente de

nuevos lenguajes a los que se denominó "de cuarta generación". Estos lenguajes tenían como objetivo obtener resultados de calidad, en forma mucho más rápida que los obtenidos con los lenguajes convencionales, (tercera generación). Algunos de estos lenguajes fueron diseñados para simplificar la construcción de ciertos tipos de aplicaciones, de manera que los usuarios finales, al igual que los que realizaban el software, pudieran construir programas y obtener los resultados esperados.

La mayoría de estos lenguajes fueron diseñados para los analistas y no para los programadores. Permitían crear pantallas, informes, diálogos, bases de datos, navegar a través de las bases de datos y actualizar los datos, todo esto sin usar técnicas de programación. También permitían traducir directamente sus diseños estructurados en el código de programación.

Este concepto de análisis de sistemas asistido por ordenador es el deseable para transformar las técnicas estructuradas y los lenguajes de programación. Marca el comienzo del diseño automático, en que se construye un lenguaje para las técnicas de diseño. Esto producirá cambios fundamentales en los lenguajes de programación, especialmente cuando limitemos el diseño a aquellas

estructuras que puedan ser verificadas en forma matemática.

3.1.5.4.- PARTICIPACION ACTIVA DEL USUARIO FINAL

En el diseño de sistemas es esencial la participación del usuario final. Algunos diseñadores y programadores tienen la tendencia a trabajar solos, obtienen diseños que tal vez son muy elegantes, pero sin la participación del usuario final. Esto es dañino y por lo general tiene como resultado el que muchos sistemas no satisfagan las necesidades reales del usuario, lo cual en muchos casos, se transforma en una gran pérdida de tiempo y dinero y en una no implantación del sistema.

Así pues,la participación del usuario final, se convierte en un principio de vital importancia en el análisis y diseño de sistemas. Esta participación se puede hacer de diferentes formas:

- Los usuarios pueden pensar, describir y discutir cuáles serían los procesos a mecanizar más usados por ellos.

- Los usuarios deben ser capaces de leer y criticar

las especificaciones creadas por el equipo encargado del análisis.

- Los usuarios deben estar capacitados para utilizar ciertas herramientas, que les permitan a ellos mismos crear sus propias aplicaciones mecanizadas.

- Los usuarios deben estar fuertemente involucrados en el proceso de administración de datos para definir y diseñar el modelo de los datos.

- Los usuarios que pertenecen a niveles de decisión, deben estar involucrados en la definición de los datos que se necesitan al más alto nivel así como en los objetivos globales del sistema.

- En algunos casos los sistemas deben ser diseñados mediante la técnica del prototipo. Esto tiene por objeto que sean probados por los usuarios antes de llegar a hacer mayores inversiones de dinero en las etapas posteriores.

- Se debe aprovechar al máximo la creatividad del usuario.

Para involucrar al usuario en el análisis, diseño y administración de datos es deseable el empleo de técnicas estructuradas lo más sencillas posible. Para esto es esencial contar con una buena técnica de diagramación. El usuario debe pensar en los sistemas ayudado de diagramas que sean claros, y pudiendo describir los procedimientos que necesita.

La diagramación se puede hacer a dos niveles. Un primer nivel puede estar formado por borradores que el usuario realiza y corrige de una manera simple, para luego pasar a un nivel más sofisticado, en el que el diseño se va refinando hasta llegar a ser más riguroso. Este diseño riguroso permitirá la verificación axiomática de sus estructuras. Esta fase ya es ejecutada por un especialista. La transición desde el borrador del usuario al diseño riguroso se puede hacer en forma natural y no tiene por qué estar sujeta a diferentes formas de lenguaje. La transición se produce por medio de una evolución, en la que se va agregando un mayor nivel de detalle a las estructuras que componen los diagramas. A su vez, estos diagramas podrán llegar a descomponerse a un nivel de detalle tal, que a partir de ellos se pueda generar el código del programa.

Resumen.-

A partir del estudio de las técnicas estructuradas, su filosofía y evolución en el tiempo, y de acuerdo a las nuevas necesidades de desarrollo que se van presentando, podemos obtener un cuadro que reune los principios antes enunciados y que son la base para el desarrollo de la metodología CASE MIDES.

PRINCIPIOS DERIVADOS DEL CONCEPTO DE MODULARIDAD

- **Principio de Localización**
- **Principio de Información Oculta**
- **Principio de Abstracción**

PRINCIPIOS DERIVADOS DE LAS TECNICAS ESTRUCTURADAS

- **Principio de Orden Jerárquico**
- **Principio de Ortogonalidad**
- **Principio de Formalidad**

PRINCIPIO DERIVADOS DEL ENTORNO DE BASES DE DATOS

- **Principio de la Independencia de los Datos**

PRINCIPIOS DERIVADOS DE LA INGENIERIA DEL SOFTWARE

- **Dirección a través de un plan de ciclo de vida de desarrollo por fases**
- **Realicaciones de validaciones**
- **Mantenimiento estricto del control del producto**
- **Uso de técnicas modernas de diseño**
- **Mantenimiento claro de responsabilidades**
- **Emplear menos gente y mejor**
- **Mantener un compromiso para mejorar el proceso**

PRINCIPIOS DERIVADOS DEL DISEÑO ASISTIDO POR ORDENADOR

- **Evitar la programación cuando sea posible**
- **Empleo de herramientas gráficas**
- **Los lenguajes deben construirse para las técnicas de diseño**
- **Participación activa del usuario final**

Figura 3.14 ***Principios de la metodología MIDES***

3.2.- TECNICAS

3.2.1.- JAD

El JAD (Joint Aplication Design) constituye una alternativa a la técnica tradicional de entrevistas en cuanto a método para analizar requisitos de sistemas. El JAD es un trabajo de dos a cuatro días en los cuales varios usuarios cualificados se reunen con personal de sistemas de información. El método está concebido para producir el mismo producto final que con los enfoques tradicionales pero en menos tiempo y con menor coste.

Con el JAD se intentan reducir los siguientes problemas que surgían con las entrevistas:

- Consumo de muchísimo tiempo y de bastantes recursos para la recogida de requisitos.

- Mal entendimiento de los requisitos por parte de los usuarios y, como consecuencia, errores en la especificación del sistema con su consecuentes costes asociados e insatisfacción por parte de los usuarios.

El JAD además promociona un estilo participativo para el desarrollo de sistemas, lo que implica una mejor comprensión del sistema por parte del usuario y fomenta un sentimiento de pertenencia al sistema.

Participantes

Usuarios:

Son los participantes clave cuyas perspectivas determinan la mayoría de los requisitos. Es importante contar con tantos usuarios como sea necesario para llegar a reunir un aspecto equilibrado de puntos de vista. Sin embargo un grupo demasiado grande es inmanejable. Es importante que los usuarios participantes tengan familiaridad con los sistemas informáticos para que haya una comunicación efectiva en las reuniones.

Analista Funcional del equipo de desarrollo:

En contraste con la técnica de entrevistas, el papel que desarrolla es pasivo, simplemente escuchando los requisitos y asegurando que son razonables y comprensibles.

Observadores:

Es adecuado que el analista funcional esté apoyado por observadores que proporcionen asistencia técnica y

consejos. Estos no tienen que ser necesariamente del departamento de informática.

Secretarios:

Lo ideal es contar con dos personas que tomen notas de todo lo que se diga en la reunión:

- uno perteneciente al departamento de desarrollo y que se haga cargo de tomar notas de los requisitos funcionales, definiciones de procesos y definición de los datos.

- otro del departamento de los futuros usuarios, que tome notas de los requisitos técnicos, volúmenes y restricciones de tiempo de realización y rendimiento.

Guía de la sesión:

Es el encargado de dirigir la reunión según los métodos del JAD. De él depende, la mayoría de las veces, el éxito de las reuniones. El objetivo que dirige la actuación del guía de la sesión es procurar la libre expresión de conocimientos y opiniones, y el fomento de un clima de consenso.

Directivo patrocinador:

Se debe contar con un directivo del departamento de los usuarios, con cierta autoridad directa o moral sobre el personal del departamento de desarrollo de sistemas informáticos. El cometido de esta persona es asegurar ante los participantes el compromiso de la alta dirección tanto con el proyecto como con el método JAD. Sólo estará presente en su inicio y en su conclusión.

El JAD ha sido utilizado con éxito en varias compañías como Texas Instruments, IBM, TWA, etc. Existen otros métodos alternativos basados en técnicas en grupo de análisis de requisitos, pero JAD se diferencia de las otras en que se centra en los flujos de trabajos, en vez de centrarse en funciones o análisis de datos, diferencia esencial para el enfoque que pretendemos dar en la metodología MIDES.

3.2.2.- DIAGRAMAS DE FLUJO DE DATOS

El Diagrama de Flujo de Datos, como hemos visto en el Estado de la Cuestión, es una técnica fundamental en las metodologías de sistemas estructurados. En estas

metodologías el DFD es usado en conjunción con otras técnicas en la fase de análisis del proceso de desarrollo.

Ya ha quedado expuesta esta técnica en el apartado 2.3.1 y las diferencias existentes entre las versiones de DeMarco, Gane y Sarson, SSADM. El Diagrama de Flujo de Datos que proponemos emplear es el de DeMarco, pero con las extensiones de notación del Esquema de Transformación de Ward y Mellor (apartado 2.3.4). De esta manera podremos especificar adecuadamente aquellos subsistemas del Sistema de Información con características de tiempo real. Si el sistema a especificar no tiene claramente características de tiempo real aconsejamos no usar la terminología extendida de Ward y Mellor, ya que lo único que lograríamos sería complicar el entendimiento del sistema.

Hemos seleccionado la técnica de DFD para la representación del modelo de comportamineto del sistema por su conformidad con los principios básicos establecidos (apartado 3.1) de la metodología que proponemos.

Principios derivados de la modularidad (apartado 3.1):

- la abstracción (apartado 3.1.1.3), donde podemos observar el sistema tanto a un nivel general como

a un nivel detallado mientras que mantengamos los enlaces y las interfaces entre los diferentes niveles (regla de balanceo).

- la localización (apartado 3.1.1.1), donde cada proceso está dirigido a especificar una característica funcional del sistema. De este modo cualquier cambio en la especificación de requisitos software del sistema puede ser fácilmente localizada y modificada.

- la información oculta (apartado 3.1.1.2), aplicando la regla de conservación de datos "*un proceso debe ser capaz de construir sus salidas usando solamente la información suministrada por los flujos de datos de entrada, más información constante*".

Principios derivados de las técnicas estructuradas

- orden jerárquico (apartado 3.1.2.1): este principio es aplicado para conseguir una mejor legibilidad y por tanto una mayor comprensión del sistema, pero carece de unas reglas estrictas para realizar el desglose por niveles del sistema. No hay que olvidar que con el conjunto de diagramas de flujo de datos

lo que se pretende es representar el sistema en forma de red, no en forma jerárquica. Esta se da para que el sistema pueda ser entendido por la dirección (la lectura queda restringida a los niveles superiores que refleja el sistema a grandes rasgos), y por otro lado los usuarios y analistas pueden leer desde lo abstracto al detalle en áreas de interés. Además esta descomposición por niveles puede ser usada como una guía para elaborar el plan de verificación y validadción.

- principio de ortogonalidad (apartado 3.1.2.3), se aplica en cuanto que con tan sólo tres elementos, ya que las entidades externas únicamente aparecen en los diagramas de contexto, se es capaz de representar el sistema completo.

- principio de formalidad (apartado 3.1.2.2), no se aplica en las metodologías actuales en cuanto que no existe un método riguroso para la creación de los diagramas de flujo de datos; pero en el apartado siguiente proponemos un procedimiento formal para su realización.

Otros principios que se aplican son:

Es gráfico (apartado 3.1.5.2), y fácil de entender y usar por los usuarios, los cuales deben participar activamente en el desarrollo del producto en esta fase (apartado 3.1.5.4). Además se consigue con ello un claro mantenimiento de responsabilidades (apartado 3.1.4.5), ya que el resultado obtenido en esta fase es de suma importancia porque constituirá lo que el sistema deberá hacer.

3.2.3.- DIAGRAMA DE OBJETIVOS

Es una técnica que se basa en un modelo para reflejar los objetivos y finalidades que la empresa asigna al sistema a desarrollar. Este modelo evolucionará cuando los objetivos de la empresa se amplíen o se modifiquen.

El conjunto de datos de un sistema informático a desarrollar comprende datos concernientes a:

- El propio Sistema.
- Las Entidades Externas que están en relación con él.
- Los intercambios de las Entidades Externas con el

Sistema Informático a desarrollar.

- Los objetos de los intercambios.

Parte del siguiente diagrama general:

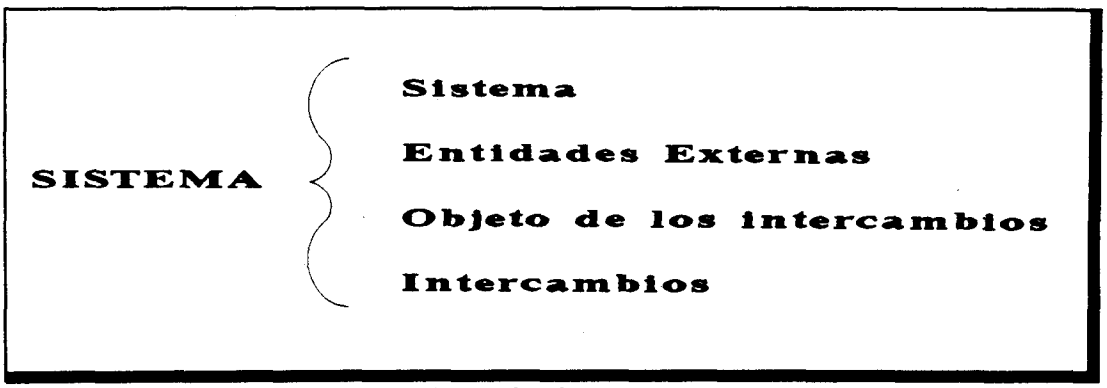


Figura 3.15 ***Diagrama de Objetivos***

Los datos concernientes a los intercambios contienen exclusivamente los intercambios de Entidades Externas con el Sistema Informático, y no conciernen nunca a los intercambios entre Entidades Externas, para los cuales el Sistema Informático no servirá de intermediario.

Puede darse que cierto conjunto de datos sea vacío en el momento de acometer el proyecto informático.

Con esta aproximación jerárquica de los datos logramos definir los continentes antes que los contenidos. El interés de esta aproximación es permitir una organización

que pueda transformarse posteriormente, haciendo que los datos puedan corresponderse con las futuras necesidades de la empresa, y de este modo posibilitar la evolución de las soluciones informáticas existentes.

Los niveles siguientes se completan en base a los objetivos.

Ejemplo:

Una empresa quiere diseñar un sistema para gestionar un pedido global a partir de los pedidos individuales realizados por los almacenes que tiene. Cada almacén genera un pedido por proveedor, si lo hace. La Central de Compras genera un pedido por proveedor. Los proveedores ofrecen tramos de descuentos por cantidad para cada producto. Todo pedido comprende una serie de productos. El almacén puede realizar independientemente otros pedidos notificando al final del pedido todo lo vendido y todo lo guardado en reserva de cada producto. Esta información histórica se guardará para la petición del año siguiente.

Aplicando estos objetivos al modelo tenemos:

Con esto tenemos un primer conjunto de objetos (figura 3.17).

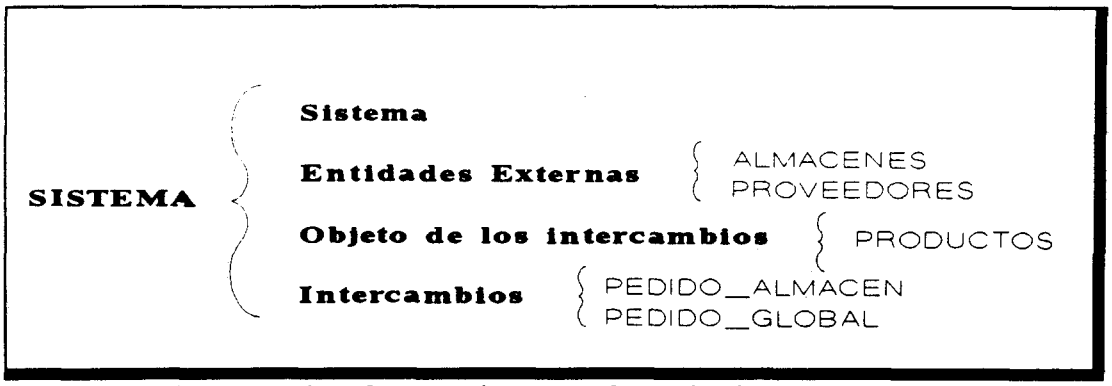


Figura 3.16 ***Ejemplo de Diagrama de Objetivos***

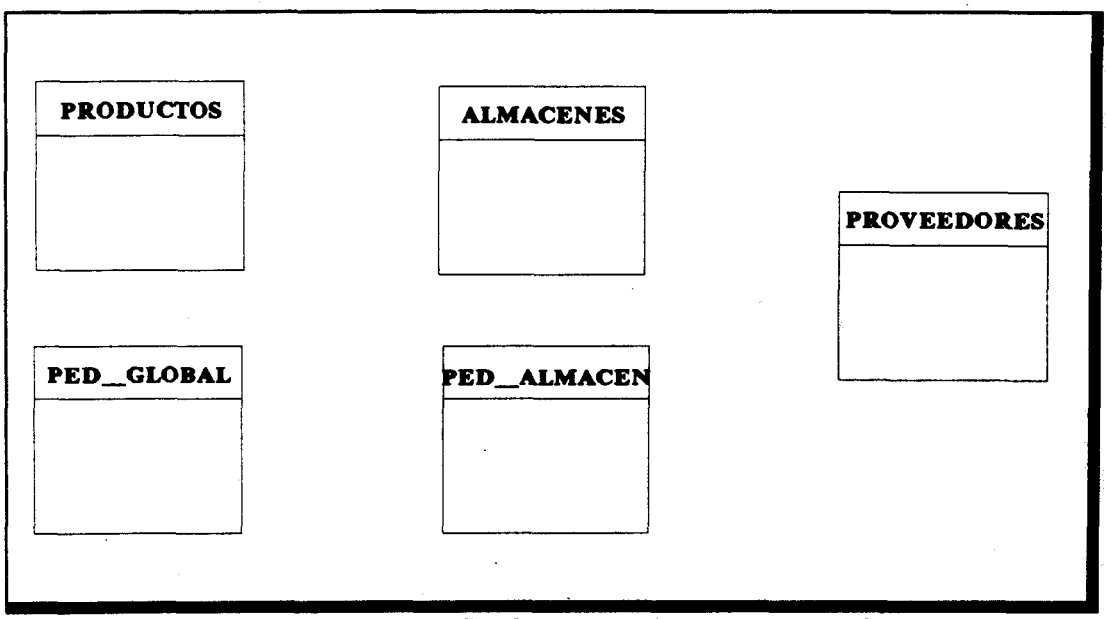


Figura 3.17 ***Objetos derivados del Diagrama de Objetivos***

Teniendo en cuenta los criterios de gestión de los objetivos, vemos que un almacén puede generar de 0 a N pedidos, luego tenemos una relación Almacén-Pedido_Almacén. A su vez se genera un pedido global por proveedor de productos, luego aparece una relación Proveedor-Pedido_Global. Asimismo tenemos que todo producto tiene una

serie de tramos de descuento; como tramo no existe se crea y se establece la relación. Revisando todas las reglas de gestión de esta manera se obtiene el siguiente modelo, (ver figura 3.18).

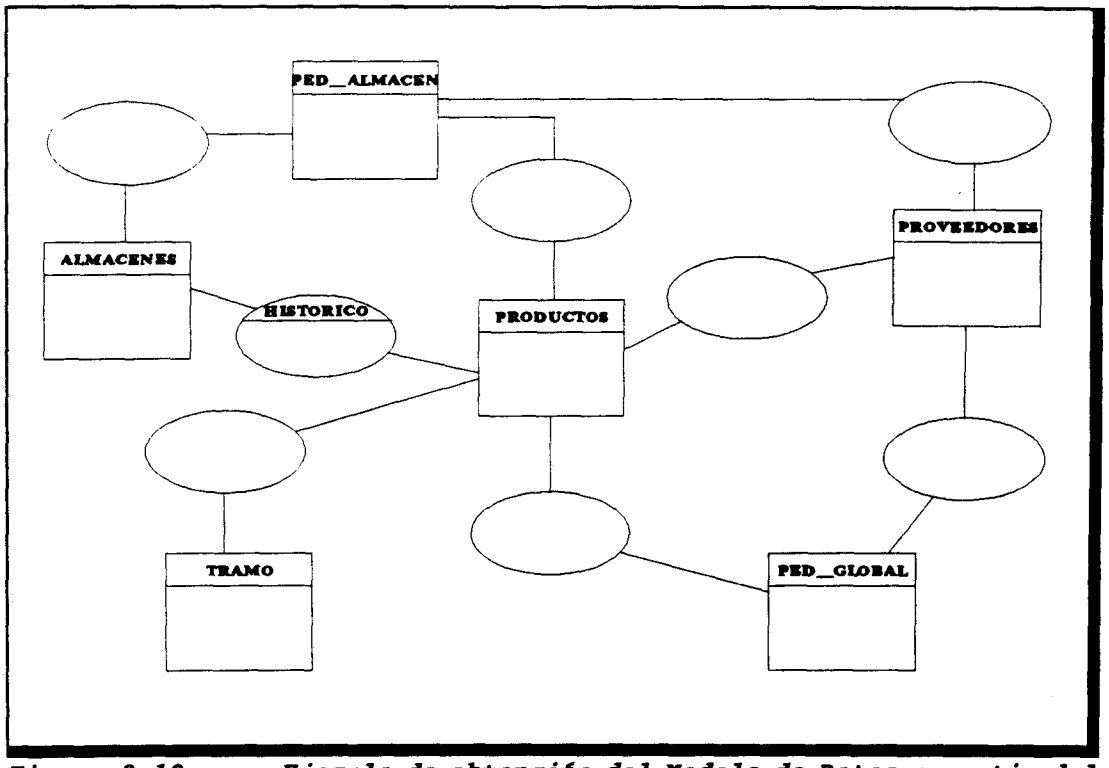


Figura 3.18 ***Ejemplo de obtención del Modelo de Datos a partir del Diagrama de Objetivos y Reglas de Gestión.***

Finalmente, se incluyen las cardinalidades deducidas de las reglas de gestión: un Pedido incluye varios Productos, un Producto pertenece a un Proveedor, un Pedido_Almacén pertenece a un Almacén y un Proveedor, etc., resultando un Modelo Conceptual de Datos, (ver figura 3.19).

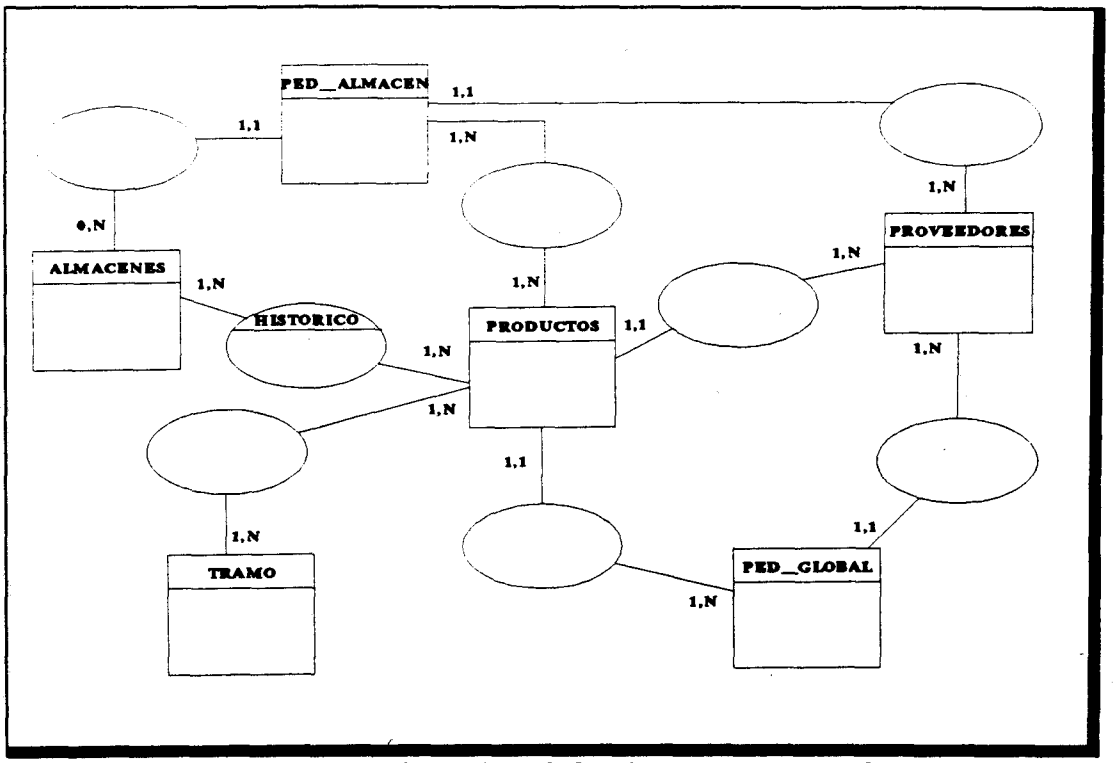


Figura 3.19 ***Obtención del Modelo de Datos a partir del Diagrama de Objetivos y Reglas de Gestión.***

Se ve cómo a partir del Diagrama de Objetivos y de las Reglas de Gestión se obtiene de forma sencilla y mediante aproximaciones sucesivas el Modelo Conceptual de Datos.

El Diagrama de Objetivos además nos servirá para determinar las Entidades Externas y los flujos de datos de entrada y salida al sistema en el Diagrama de Contexto.

3.2.4.- WALKTHROUGH

La técnica de Walkthrough o Revisión en Equipo [AEC86] consiste en la revisión de cualquier producto software por un grupo de personas cuyo nivel profesional es similar al de la persona o el grupo de personas que desarrolló dicho producto. La razón por la que incluimos esta técnica en la metodología MIDES es que es un método efectivo para conseguir la calidad en los productos software [YOU89].

El Walkthrough propuesto es del tipo semiformal, por tanto auna características de los Walkthrough formales e informales de la siguiente manera:

Formalidad

- Se establecen una agenda y unos objetivos.
- Se cuenta con un límite para el tiempo de duración.
- Existen procedimientos, de común acuerdo, sobre cómo debe desarrollarse la reunión.

Informalidad

- Sólo interviene personal del mismo nivel que el del autor del producto.

- Se compone casi enteramente de personas directamente involucradas en el proyecto, a tiempo completo (no parcial).

FUNCIONES

Administrativas y organizativas

Presentador:
Es el autor del producto y lo presenta al grupo que lo va a revisar.

Coordinador:
Asegura que las actividades se planifican y organizan adecuadamente. Durante la reunión actúa como un moderador.

Secretario:
Toma notas durante la reunión, que formarán el registro de los resultados de la misma. Deberá poseer los conocimientos necesarios para poder sintetizar los comentarios vertidos en la reunión.

Revisión

Oráculo de mantenimiento:

Revisa el producto desde el punto de vista de su mantenimiento futuro. Deberá tratar de predecir los cambios que deberán hacerse al producto en el futuro.

Supervisor de estándares:

Se preocupa de que el producto muestre su conformidad con los estándares aplicables.

Representante del usuario:

Asegura que el producto satisfaga las necesidades del usuario, evitando el fenómeno lamentable de crear una solución brillante para el problema equivocado.

Otros revisores:

Su objetivo primordial es contribuir con una opinión general sobre la corrección y la calidad del producto.

El responsable de elegir a los participantes del Walkthrough será el equipo de desarrollo del proyecto al completo. El número de personas a participar oscilará entre cuatro y seis, ya que más pueden complicar las reuniones y convertirlas en comités ineficaces.

3.2.5.- MATRIZ DE ACCIONES

Sea A_i una acción recogida a través del proceso de determinación de las especificaciones. Representaremos por ΣA_i el conjunto de acciones correspondientes a las especificaciones de un sistema. Análogamente E_i corresponderá a un evento del sistema y ΣE_i el conjunto de eventos del sistema.

Los componentes de esta matriz C_{ij} contendrán:

- **un cero (0)**: si el evento E_i no participa en la acción A_j.
- **un uno (1)**: si el evento E_i participa como entrada solamente.
- **un menos uno (-1)**: si el evento E_i participa como salida.

La figura 3.20 muestra una Matriz de Proceso donde (A_1 A_2 ... A_j ... A_m) son las acciones involucradas en el problema, (E_1 E_2 ... E_i ... E_n) son los eventos involucrados en el problema y (C_{11} ... C_{ij} ... C_{nm}) son los usos de cada evento en cada acción.

Aquellas filas que sólo tengan algún 1 con o sin

ceros, son eventos iniciales (externos) o sea de entrada al sistema y por tanto son los requeridos por el modelo de datos. Los llamaremos **eventos rígidos**. Aquellas filas que sólo tengan -1 con o sin ceros, son eventos finales resultados. En el caso de que una fila tenga unos (1) y menos uno (-1) en una fila corresponde a eventos intermedios resultados de unas acciones y utilizadas por otras.

	A_1	A_2	$\cdots$	A_j	$\cdots$	A_m
E_1	C_{11}	C_{12}		C_{1j}		C_{1m}
E_2	C_{21}	C_{22}		C_{2j}		C_{2m}
$\vdots$						
E_i	C_{i1}	C_{i2}		C_{ij}		C_{im}
$\vdots$						
E_n	C_{n1}	C_{n2}		C_{nj}		C_{nm}

Figura 3.20 Matriz de Acciones General

Ejemplo:

E1 y E2 son eventos externos. E6 es un resultado y el resto son eventos intermedios. Representado en forma de grafo, sería el de la figura 3.22.

A partir de la Matriz de Acciones se puede generar en primer lugar las acciones en operaciones. Se entiende por operación el conjunto de acciones que se realizan a partir de los mismos acontecimientos externos.

	A_1	A_2	A_3	A_4
E_1	1	1	0	0
E_2	0	1	1	0
E_3	0	−1	1	0
E_4	0	0	−1	1
E_5	−1	0	0	1
E_6	0	0	0	−1

Figura 3.21 ***Matriz de Acciones***

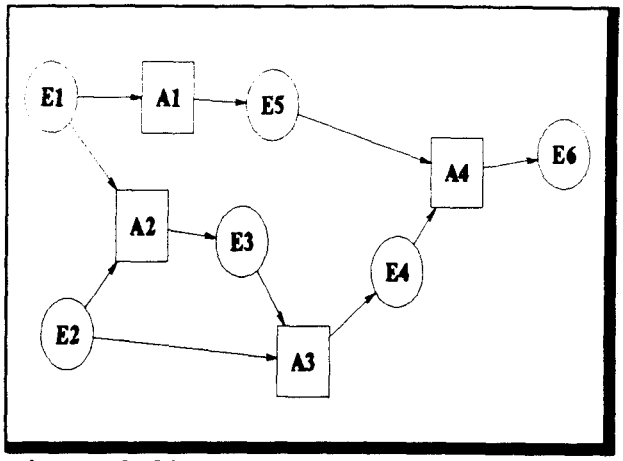


Figura 3.22 ***Grafo de la Matriz de Acciones***

Bastará para ello desarrollar un algoritmo que agrupe las acciones que requieran los mismos eventos rígidos, o que estén intercomunicados, (que el evento o eventos entrada en la acción sean la salida de acciones encuadradas en la operación).

Ejemplo 1: Sea el grafo de acciones (figura 3.23).

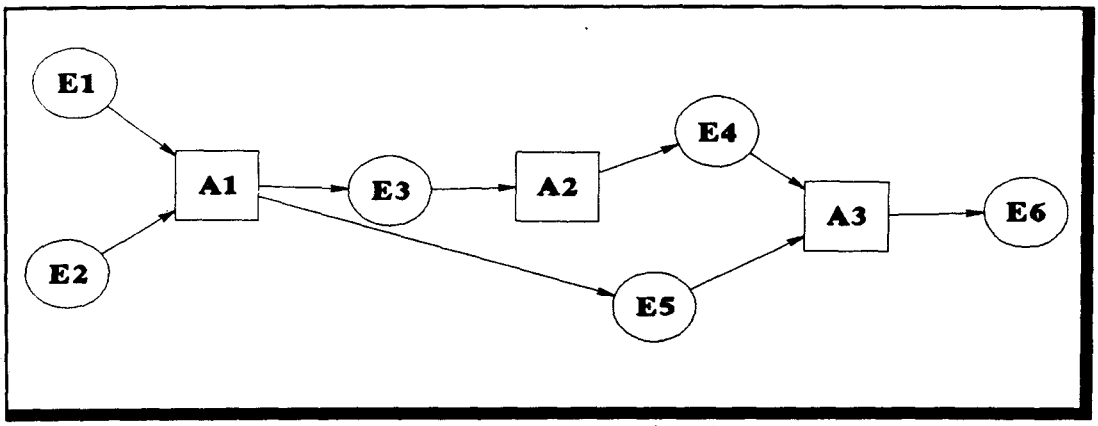


Figura 3.23 ***Grafo de acciones del ejemplo 1***

Este ejemplo corresponde a una operación. E1 y E2 son eventos iniciales de A1, E3 evento entrada a A2 y salida de A1, y E4 y E5 eventos entrada en A3 procedentes ya de las acciones A1 y A2 incorporadas ya en la operación. Luego en este caso se ve claramente que todas las acciones están constituyendo una operación.

Ejemplo 2: Sea el grafo de acciones (figura 3.25).

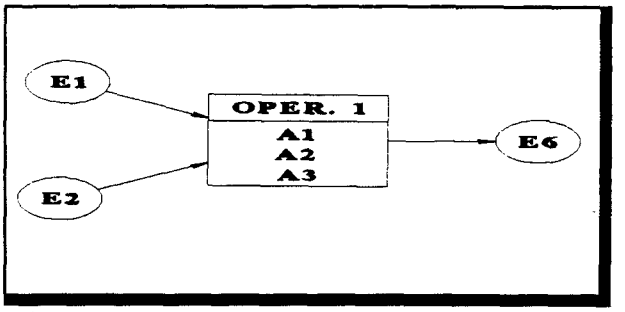


Figura 3.24 *Grafo de Operaciones (ejemplo 1)*

En este ejemplo E1 y E2 son eventos iniciales de A1, E3 evento entrada a A2 y salida de A1 luego pertenece a la misma operación, y finalmente A3 requiere un evento externo distinto, luego

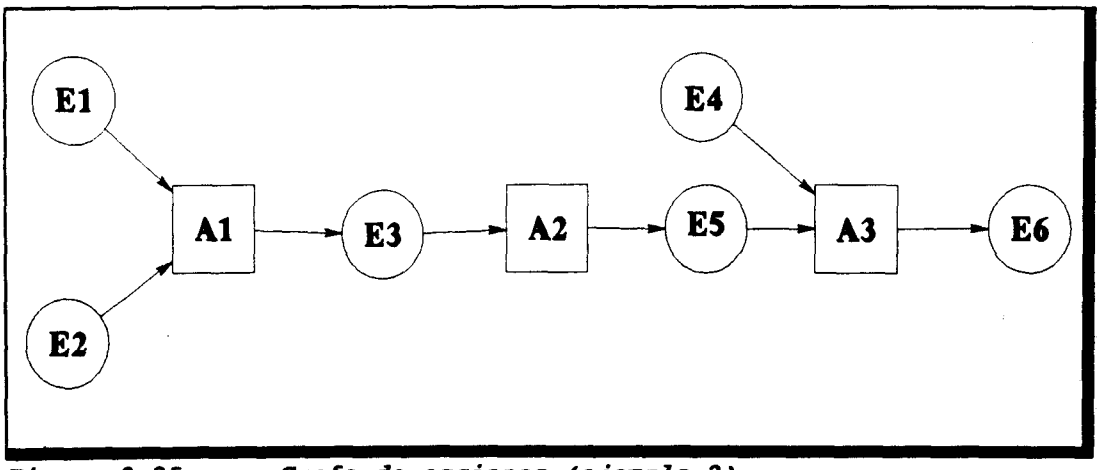


Figura 3.25 *Grafo de acciones (ejemplo 2)*

pertenece a una segunda operación. El grafo de representación sería:

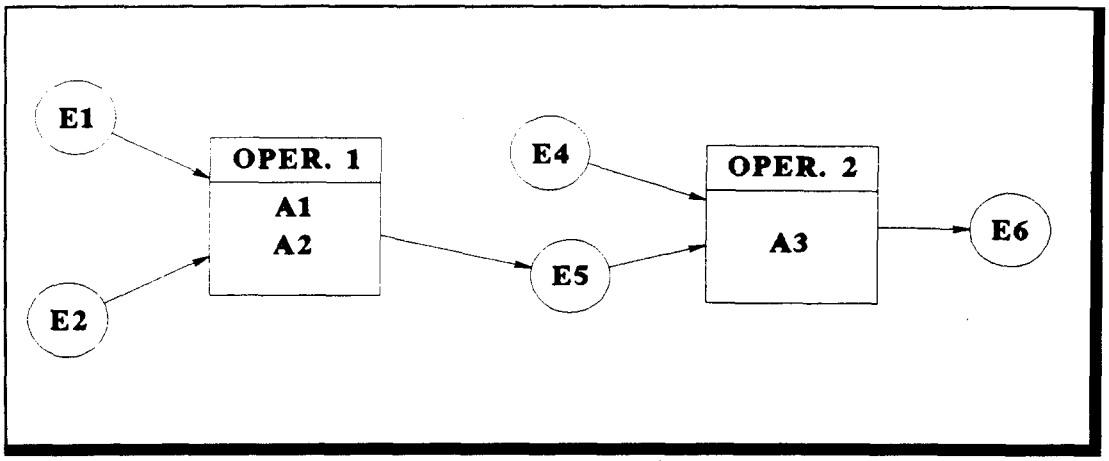


Figura 3.26 ***Grafo de Operaciones (ejemplo 2)***

3.2.6.- MATRIZ DE OPERACIONES/EVENTOS

El objetivo de esta matriz es el de determinar los procesos de que consta el sistema. Llamamos **proceso** a un conjunto de operaciones interrelacionadas.

Este problema tiene gran interés para la determinación del nuevo modelo físico del sistema.

Las filas de esta matriz las constituyen el conjunto de operaciones de que consta el sistema, y las columnas por los eventos de entrada y salida de cada operación.

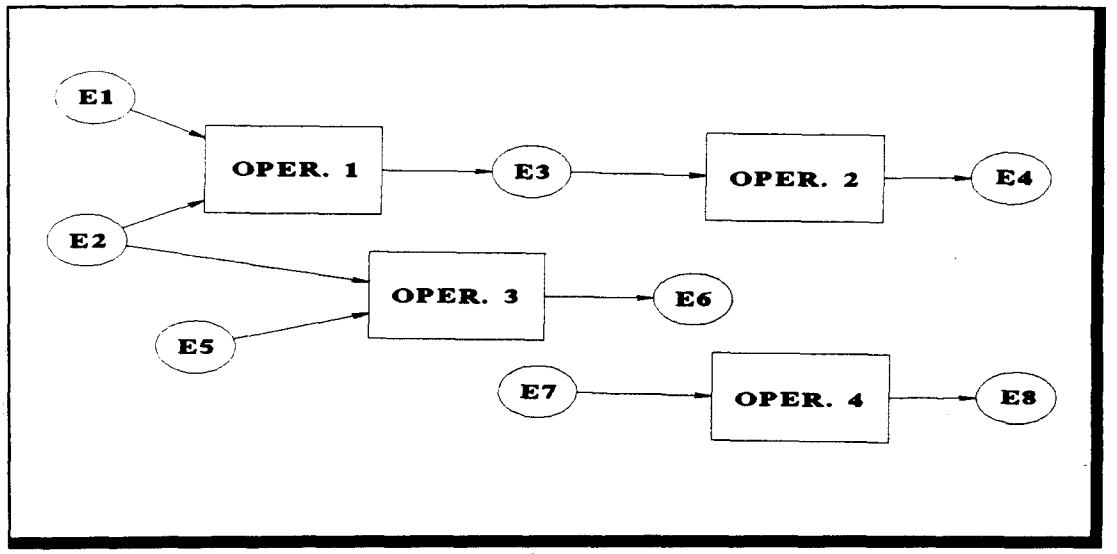


Figura 3.27 ***Grafo de Operaciones***

Los componentes de esta matriz contendrán una **X** si el evento participa tanto de la entrada como de la salida a la operación.

Presentamos un ejemplo teórico para exponer este objetivo ya que un caso concreto puede requerir un desarrollo demasiado largo y no mejora la comprensión. Sea el grafo de la figura 3.27.

La Matriz de Operaciones/Eventos correspondiente será por tanto la de la figura 3.28; y los agrupamientos de las operaciones en procesos se realizan de la siguiente manera,

(figura 3.29):

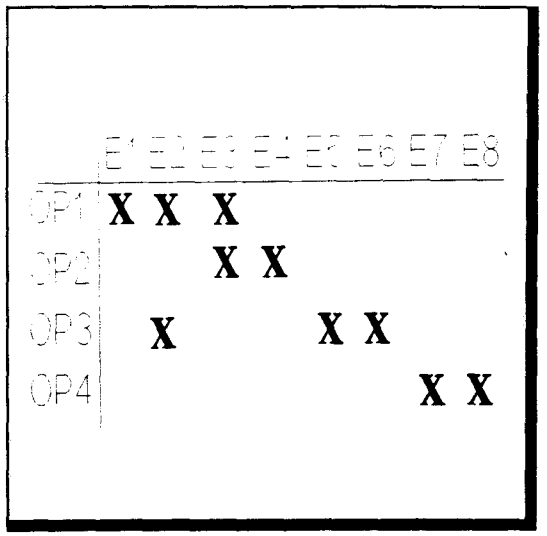


Fig. 3.28 Matriz de Operaciones Eventos

- enlazar las cruces que se encuentran en la misma fila
- enlazar las cruces que se encuentran en la misma columna

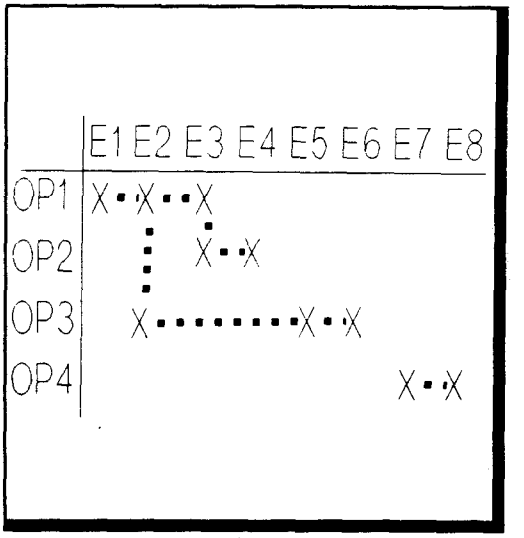


Fig. 3.29 Matriz de Operaciones Eventos conectada

Aquellas operaciones que hayan resultado enlazadas constituyen los procesos del sistema (figura 3.30).

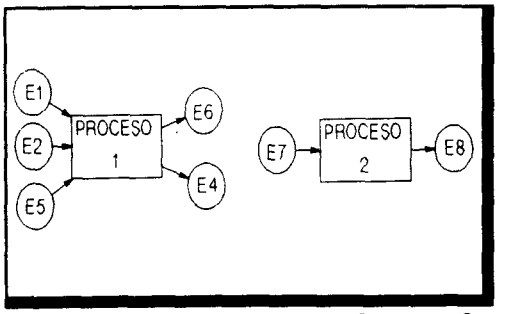


Figura 3.30 Grafo de Procesos

En conclusión lo que hemos obtenido es el conjunto de procesos totalmente independientes de que consta el sistema. Esto nos es de gran valor porque cada uno de estos procesos pueden ser

diseñados y posteriormente probados por equipos de desarrollo independientes.

3.2.7.- MATRIZ DE PRECEDENCIAS

Tiene por objeto establecer la secuencia de ejecución de las operaciones de un proceso.

Para ello en esta matriz se colocan en horizontal las operaciones del proceso y en columnas los eventos del proceso, codificándose la matriz como se muestra en el figura 3.31; es decir, se subdividen los eventos en internos y externos. Los primeros son aquellos generados por alguna operación y los segundos son los que provienen del exterior, no siendo generados por ninguna operación. Como se ve en el gráfico, con respecto a los eventos externos se pone una X si este evento participa en la operación y nada en caso contrario. Con respecto a los eventos internos colocamos un 1 en la operación que lo genera y -1 en la que lo requiere.

Asimismo se establece un vector de estado con tantos componentes como eventos internos inicializado a ceros, el cual nos indica los eventos disponibles, en cuyo caso estos

Eventos / Operaciones	Externos				Internos				
	E1	E2	E5	E7	E3	E4	E6	E8	E9
OPERACION 1	X	X			1	1			
OPERACION 2			X			-1	1		
OPERACION 3				X				1	
OPERACION 4								-1	1

Figura 3.31 ***Matriz de Precedencias***

se encuentran a uno.

El algoritmo de proceso es el siguiente:

1.- Analizar todas las operaciones de 1 a m (m = número de operaciones).

2.- 2.1 Para cada operación ver si requiere eventos internos, de no requerirlos ésta es una operación inicial.

2.2 Si los requieren, ver el vector de estado. Si éste tiene ceros en los eventos requeridos se ignora y se pasa a la siguiente operación.
Si tiene unos se conecta esta operación con la

operación que tiene estos eventos como de salida.

3.- Se repite el proceso hasta que todas las operaciones queden asignadas.

Ejemplo: Tenemos

OPERACION 1 solo requiere eventos externos y

OPERACION 3 solo genera eventos externos

Asimismo inicialmente el Vector de Estado

E4	**E8**	**E3**	**E6**	**E9**
0	**0**	**0**	**0**	**0**

Luego solo construimos OP1 y OP3 y en el Vector de Estado ponemos 1 en E3, E4 y E8:

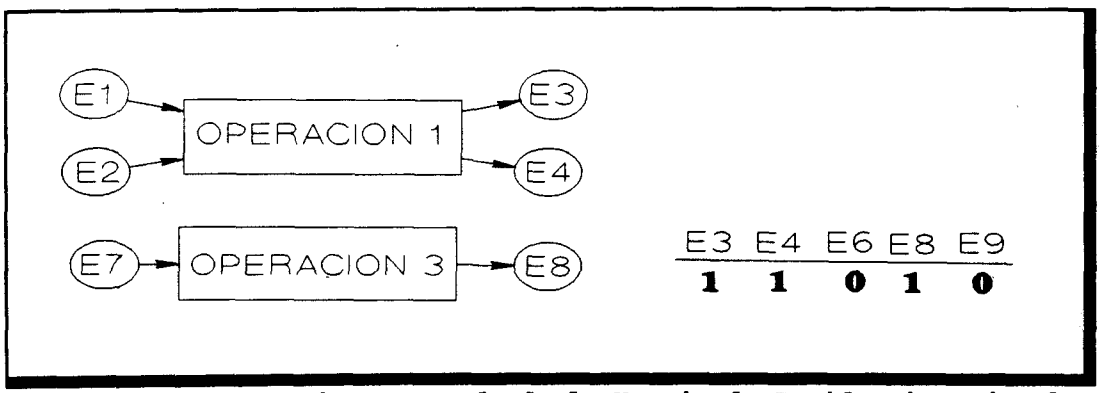


Figura 3.32 ***Primera parada de la Matriz de Incidencia (ejemplo)***

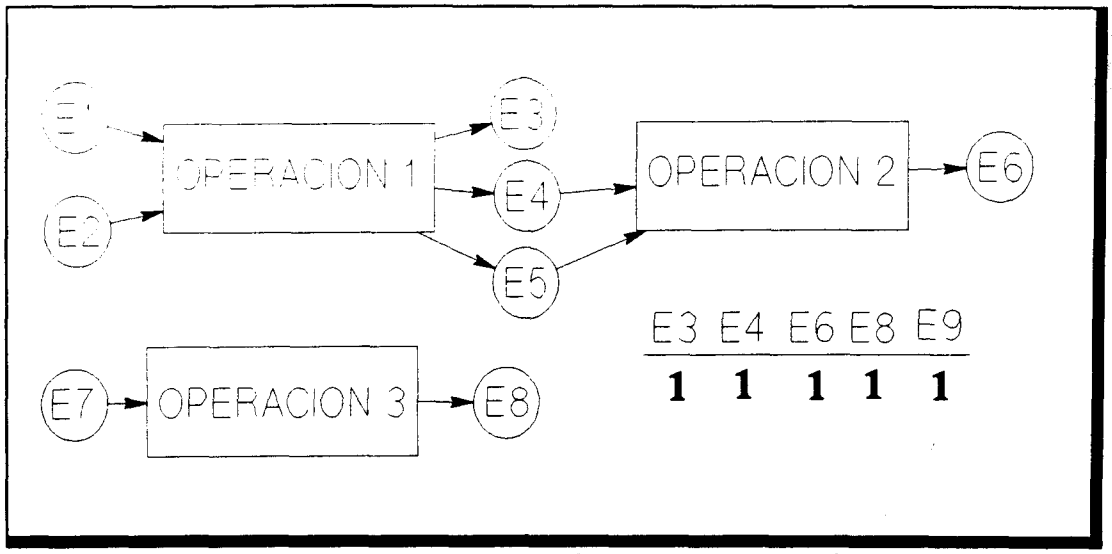


Figura 3.33 ***Segunda parada de la Matriz de Incidencia (ejemplo)***

En la segunda parada da como resultado la figura 3.33, y como están asignadas todas las operaciones se concluye el proceso.

3.2.8.- MATRIZ DE NAVEGACION

Se trata de determinar la existencia de una navegación sin ambigüedad semántica que relacione las entidades que contienen los datos requeridos por un determinado proceso.

Previamente a la determinación de la existencia de dicha navegabilidad se muestran las condiciones de viabilidad de los accesos. Las relaciones entre cada dos entidades contiguas cualesquiera pueden ser:

1.- "una a una" entre todas ellas.

2.- "una a n" entre la entidad origen y la adyacente y "una a una" todas las demás a partir de ésta.

2.- "una a n" entre entidades que no corresponden a la entidad origen.

Relación "una a una" entre todas las entidades contiguas.

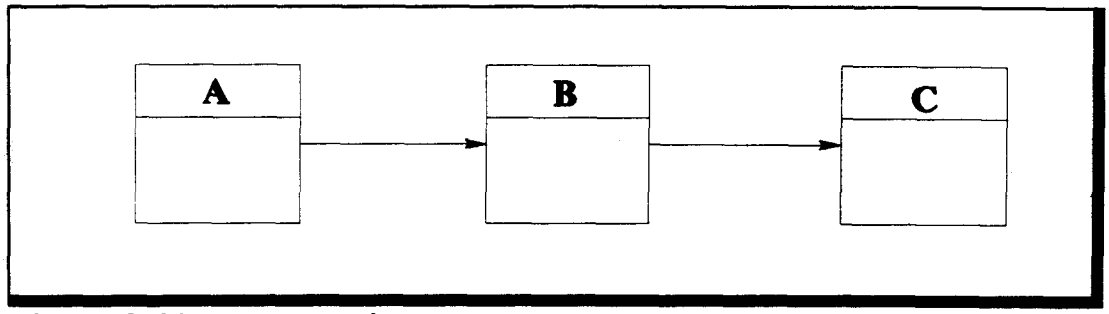


Figura 3.34 Caso 1

En este caso, en el cual existe aplicación en el sentido matemático de la teoría de conjuntos entre todas las entidades contiguas, hay por tanto una correspondencia unívoca en dicha navegación y por consiguiente no existen problemas de incertidumbre.

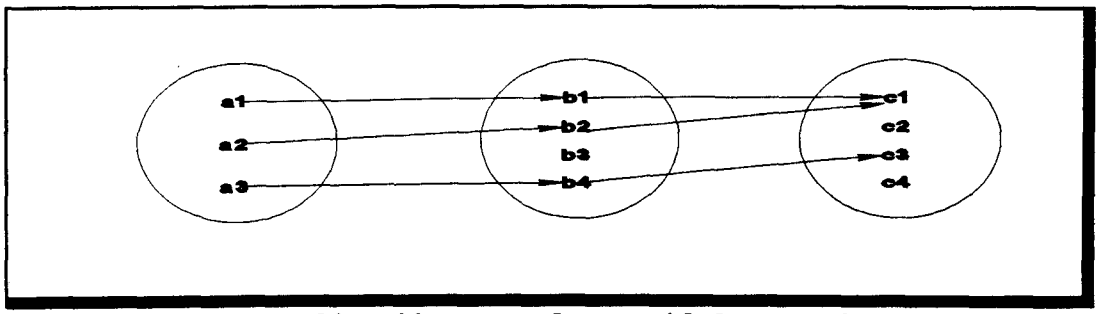


Figura 3.35 Aplicación entre las Entidades contiguas

Ejemplo:

Supongamos que cada ocurrencia de la entidad EMPLEADO, trabaje en una ocurrencia de la entidad DEPARTAMENTO; supongamos asimismo que toda ocurrencia de la entidad DEPARTAMENTO está localizada en una ocurrencia de la entidad LOCALIZACION.

El proceso tiene como objetivo dar a conocer la localización de cada empleado (figura 3.36):

En este caso tendríamos la siguiente navegación (figura 3.37).

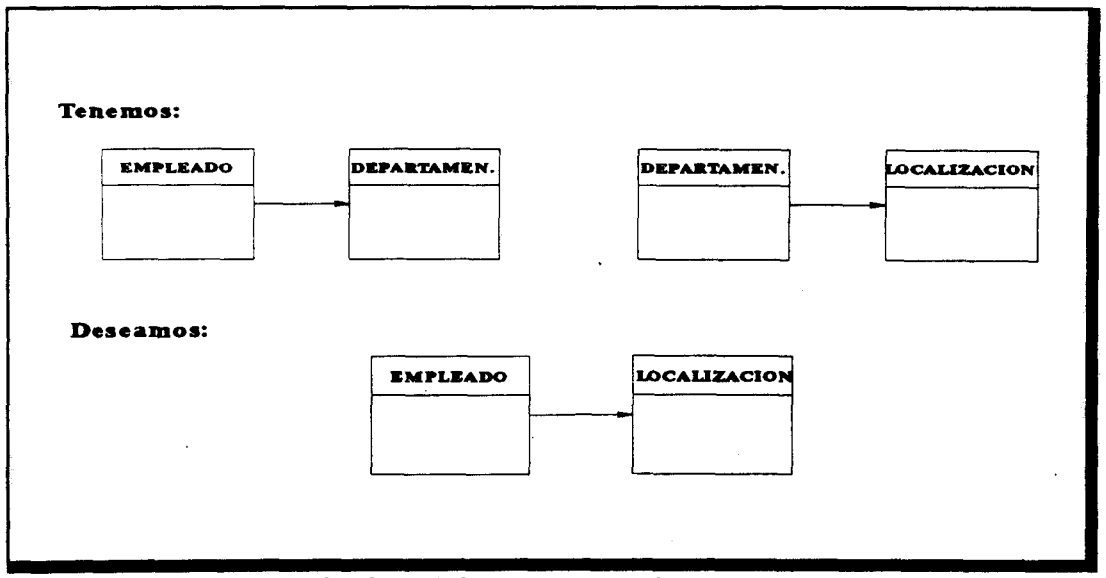


Figura 3.36 Objetivo del proceso (ejemplo caso 1)

Se ve como el EMPLEADO determina unívocamente el DEPARTAMENTO y éste a su vez determina unívocamente la

LOCALIZACION, luego por transitividad tenemos que el EMPLEADO determina la LOCALIZACION.

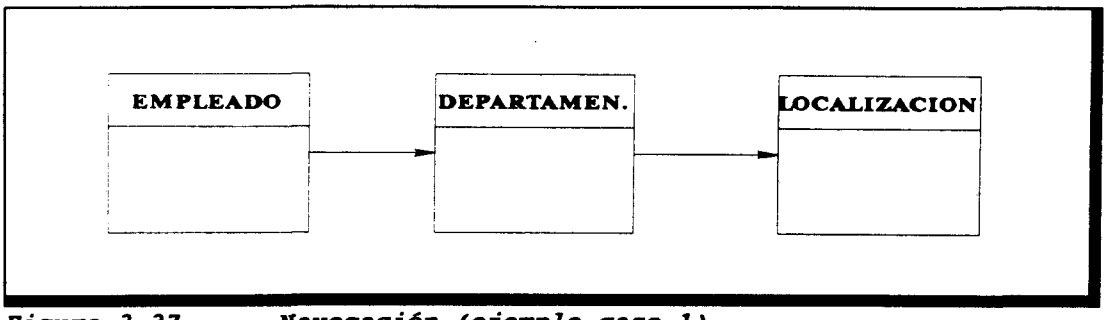


Figura 3.37 Navegación (ejemplo caso 1)

E# ----> D#

D# ----> L#

luego

E# ----> L#

Los atributos de las entidades EMPLEADO y DEPARTAMENTO que permiten estas aplicaciones se denominan criterios de aplicación.

Relación "una a n" entre la entidad origen y la contigua, y "una a una" todas las demás a partir de ésta.

Tenemos en este caso n ocurrencias de X para una ocurrencia de A, y a su vez a toda ocurrencia de X corresponde una de Y, y a una de Y una de Z, por consiguiente quedan perfectamente determinadas las

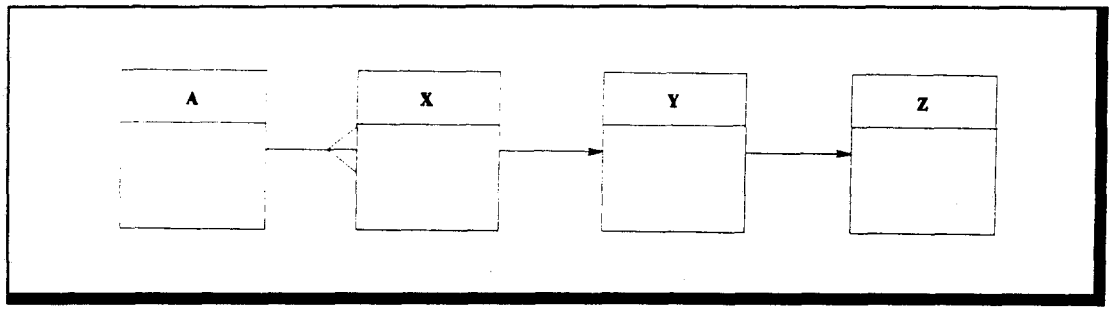


Figura 3.38 Caso 2

ocurrencias (X,Y,Z) que corresponden a cada ocurrencia de A.

Ejemplo:

Un almacén tiene un cierto número de piezas, pudiendo existir una misma pieza en varios almacenes y correspondiendo cada pieza a un único proveedor.

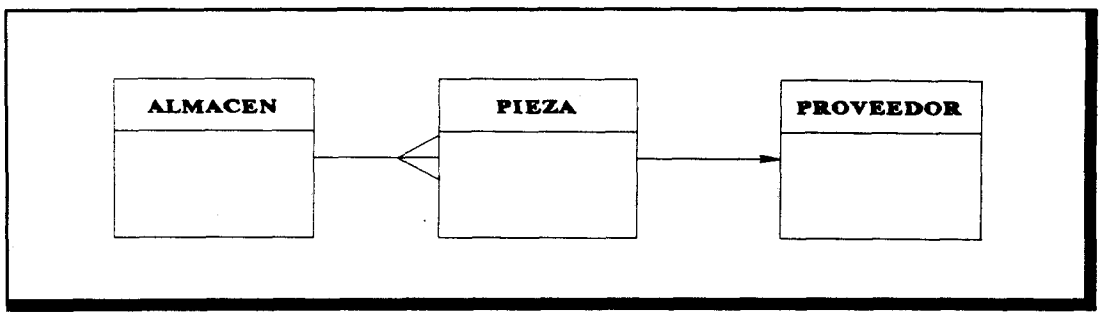


Figura 3.39 Ejemplo caso 2

Si en este modelo el proceso debe proporcionar la información relativa a los nombres de los proveedores de las piezas de un determinado almacén, el modelo proporciona a través de la primera relación todas las piezas del almacén y la segunda nos proporciona los nombres de los

proveedores, obteniéndose la información completamente y sin ambigüedad.

Relaciones "una a una" entre entidades que no corresponden a la entidad origen.

Esta situación se refleja en la figura 3.40.

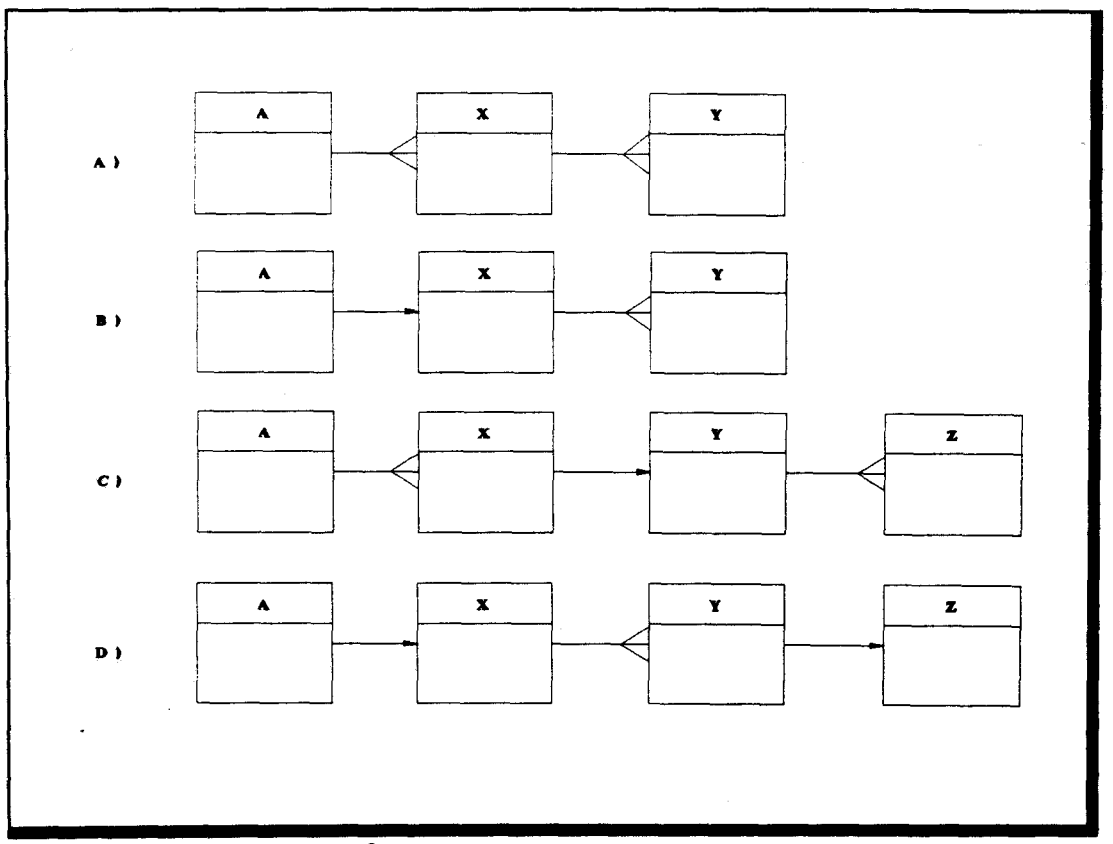


Figura 3.40 Caso 3

Considérese el caso anterior en que todo empleado

pertenece a un departamento, pero ahora el departamento puede estar ubicado en n localizaciones distintas. En este caso se tendrá (figura 3.41a)

En este caso la información es ambigüa porque el empleado tiene una localización mientras que el modelo proporciona varias, las del departamento pero no la del empleado.

Para solucionar este problema y hacer el modelo de datos compatible con el de proceso de la información sería necesario modificar el modelo de datos añadiendo la relación mostrada en la figura 3.41b.

Considérese ahora el ejemplo de los almacenes pero modificado, en el sentido que una pieza puede ser suministrada por varios proveedores (figura 3.41c).

Si de nuevo se quiere que el proceso proporcione la información relativa a los proveedores de los productos que hay en un determinado almacén, existe ahora una laguna o incertidumbre porque no se puede afirmar con este modelo qué proveedor de los que suministra una pieza, es el de dicha pieza en un almacén.

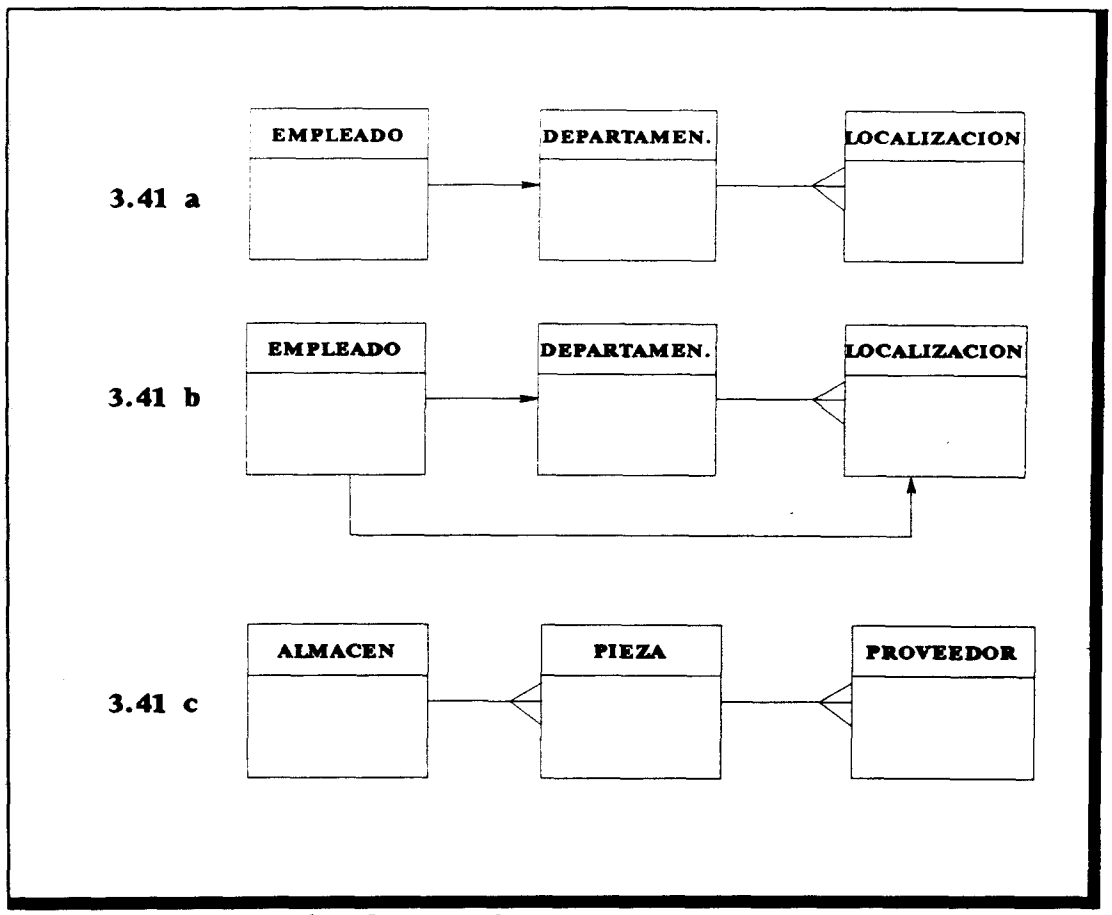


Figura 3.41 Ejemplo caso 3

Por tanto la navegación viable que determina sin ambigüedad las ocurrencias de las entidades relacionadas es la que se obtiene en los dos primeros casos.

Determinación de la viabilidad de las relaciones

Se ha visto que para que la navegación que proporcionan las entidades relacionadas sea válida, tiene que existir una relación "1 a 1" en todos los tramos del

camino, excepto en el primero que puede ser "1 a 1" o "1 a n".

Teniendo en cuenta lo anterior se establece una matriz de incidencia de entidades, la cual recoge todas las aplicaciones existentes en el modelo de datos. En esta matriz [A] cada elemento es:

$a_{ij} = 1$ si la entidad I ----> J

y $a_{ij} = 0$ si la entidad I --/-> J

Aplicando el cierre transitivo a esta matriz se obtienen todas las entidades relacionadas por transitividad como consecuencia de las aplicaciones existentes en el modelo.

En el caso de no satisfacer los requerimientos de relaciones del proceso se crea una segunda matriz de relaciones múltiples.

Seguidamente se analiza para cada entidad requerida por el proceso si ésta está en la matriz de relaciones múltiples, completando los conjuntos de entidades relacionadas uniendo a cada subconjunto resultante de dos entidades de la matriz de transitividad relacionadas con

la segunda de cada par.

Para la obtención del cierre transitivo hemos utilizado un algoritmo debido a Warshall el cual no es más que la iteración de unos algoritmos $ß_i$. Cada $ß_i$ consiste en reproducir todos los unos que existen en la línea "i" sobre toda la línea "h" que tenga un uno en la columna "i".

Si hay un uno en el elemento a_{hi} quiere decir (figura 3.42a). Si ponemos todos los unos de la fila "i" en la fila "h" estaremos creando los enlaces punteados de la figura 3.42b.

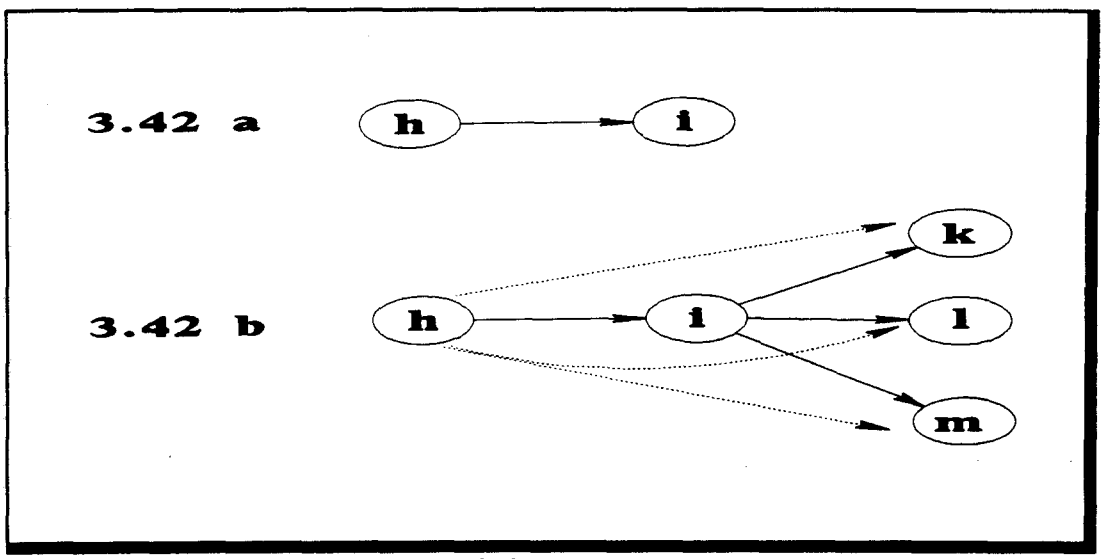


Figura 3.42 ***Cierre Transitivo***

Luego eso es lo que pretendemos, pues todas las filas que tengan un uno en la columna i representan los precedentes del nodo i-ésimo, y entonces la aplicación de β_i significa lo ya visto al definir β_i.

El algoritmo de Warshall, puesto en seudocódigo quedaría:

```
Comienzo:
   Para i=1 Hasta n Hacer
   Para h=1 Hasta n Hacer
   Si a(h,i)= 1 Entonces
      Para l=1 Hasta n Hacer
      Si a(i,l) = 1 Entonces
           a(h,l) = a(i,l)
```

Ejemplo: Sea el modelo (figura 3.43)

Este cierre transitivo proporciona los siguientes subconjuntos de relaciones:

A, C, D, E

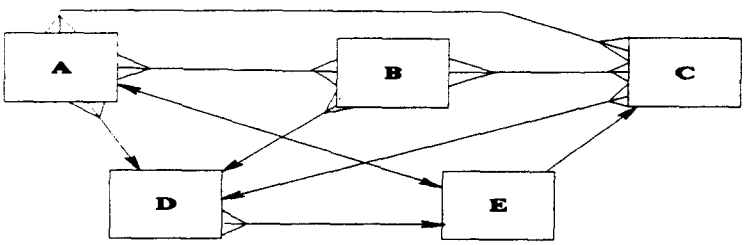
B, A, C, D, E

C, A, D, E

D, A, C, E

E, A, C, D

Como la matriz de relación múltiple proporciona los siguientes subconjuntos de pares de entidades:



	A	B	C	D	E
A	0	0	0	1	1
B	0	0	0	1	0
C	0	0	0	1	0
D	0	0	0	0	1
E	1	0	1	0	0

MATRIZ DE INCIDENCIA

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	0
C	1	1	0	0	1
D	1	1	1	0	0
E	0	0	0	1	0

MATRIZ DE RELACION MULTIPLE

	A	B	C	D	E
A	(1)	0	(1)	1	1
B	(1)	0	(1)	1	(1)
C	(1)	0	(1)	1	(1)
D	(1)	0	(1)	(1)	(1)
E	1	0	1	(1)	(1)

CIERRE TRANSITIVO

Figura 3.43 ***Matriz de Navegación (ejemplo)***

A, B

B, A B, C

C, A C, B C, E

D, A D, B D, C

E, D

resultan finalmente los siguientes subconjuntos de entidades relacionadas:

A, B, C, D, E

A, C, D, E

C, B, A, D, E

B, A, C, D, E

C, A, D, E

D, A, C, E

E, A, C, D

Todo lo anterior en definitiva nos conduce al siguiente algoritmo:

ALGORITMO PARA LA VALIDACION PROCESOS / MODELO DE DATOS

PASO 1: DETERMINAR LAS ENTIDADES REQUERIDAS POR EL PROCESO

PASO 2: ESTABLECER MATRIZ DE INCIDENCIA

PASO 3: SI MATRIZ INCIDENCIA PROPORCIONA RELACION ENTIDADES PROCESO, IR A PASO 9

PASO 4: SINO OBTENER CIERRE TRANSITIVO

PASO 5: SI MATRIZ TRANSITIVIDAD PROPORCIONA RELACION ENTIDADES REQUERIDAS POR EL PROCESO, IR A PASO 9

PASO 6: SINO OBTENER MATRIZ RELACIONES MULTIPLES Y CONCATENAR CON CADA PAREJA DE ENTIDADES DE ESTA MATRIZ, LOS SUBCONJUNTOS DE ENTIDADES DE LA MATRIZ DE TRANSITIVIDAD CUYO ORIGEN SEA LA SEGUNDA DE ESTA MATRIZ

PASO 7: SI ALGUNO DE ESTOS SUBCONJUNTOS CORRESPONDE A LA RELACION DE ENTIDADES REQUERIDAS POR EL PROCESO, IR AL PASO 9

PASO 8: SINO, MOELO DE DATOS NO VALIDO, IR AL PASO 10

PASO 9: MODELO DE DATOS VALIDO

PASO 10: FINAL VALIDACION DEL PROCESO

Figura 3.44 *Algoritmo para la Validación Procesos / Modelo de Datos*

3.2.9.- TABLA DE MULTIPLICACION

La tabla de multiplicación [VEG85] es una técnica esencial para el diseño óptimo (desde el punto de vista de ocupación de memoria e instrucciones ejecutadas) de una Acción en particular.

Supongamos que tenemos una Acción determinada en la que es preciso ejecutar unas Funciones F_1, F_2,...,F_m que operan sobre unos conjuntos de datos o variables V_1, V_2,...,V_n pertenecientes todas a un mismo conjunto referencial R; las variables se suponen de carácter aleatorio, de modo que pueden ser utilizadas "0 ó 1 veces" cada una. Además, las funciones son productos booleanos de variables o de sus complementos en R, con la particularidad de que ninguna variable figura en un producto más de una vez ni tampoco simultáneamente con su complemento. Es decir, toda función F_i (i=1,...,m) viene dada por una fórmula de tipo

$$F_i = U_{i1}.U_{i2}...U_{ik} = \prod_{p=1}^{k} U_{ip},$$

donde k <= n y para todo p existe un índice q_i (q_i= 1,...,n) tal que $U_{ip} = V_{qi}$ ó $U_{ip} = \overline{V}_{qi}$; además, todos los

factores son distintos y no puede suceder que uno sea igual a una de las variables y otro al complemento de esa misma variable.

Entre todas las soluciones que admite un problema de este tipo se trata de hallar la solución óptima.

Warnier [WAR79] da solución a este problema utilizando las tablas de Veitch, pero este procedimiento sólo es de utilidad real si el número de variables es reducido. Por el contrario en [VEG85] se llega a proponer un algoritmo para su resolución que elimina la restricción del número de variables y obtiene resultados similares.

Pasos a realizar:

1.- Ordenación de las funciones.

Diremos que una función es de altitud k si viene dada como el producto de k variables. Aceptaremos siempre que las funciones estén ordenadas de menor a mayor altitud; si hay funciones de una misma altitud se pueden ordenar de cualquier forma. Por consiguiente, la función 1 es una de las que comprenden el menor número de variables.

2.- Tabla de codificación.

Esta tabla revela la relación entre las funciones y las variables y sirve de base para las construcciones posteriores. Su estructura se muestra en la figura 3.45.

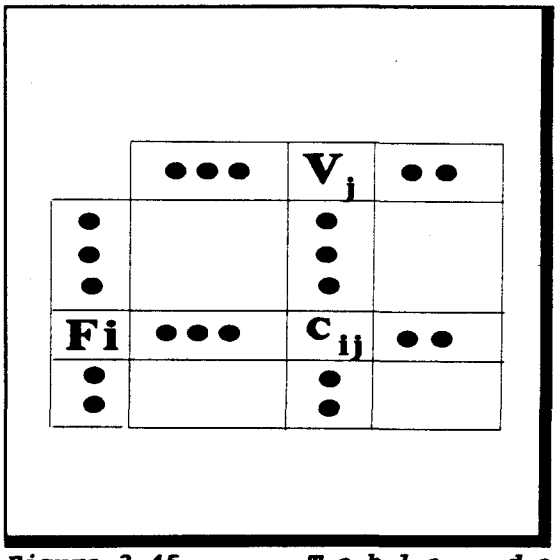


Figura 3.45 *Tabla de Codificación*

Cada columna corresponde a una variable y la asignación de las variables a las columnas es arbitraria; cada fila está vinculada a una función y si éstas se han ordenado según el criterio del punto anterior, la i-ésima fila corresponde a la i-ésima función F_i. Los elementos de la tabla de codificación son designados por c_{ij} y pueden tomar solamente uno de los tres valores 0, 1 ó X, según la variable V_j o su complemento $\overline{V}_j$ figure o no en el producto mediante el cual viene definida la función F_i; concretamente,

$$c_{ij} = 1 \text{ si } V_j \in F_i$$

$$c_{ij} = 0 \text{ si } V_j \in F_i$$
$$c_{ij} = X \text{ si } V_j, \bar{V}_j \in F_i$$

De este modo la tabla de codificación permite relacionar con toda función un vector de n componentes ternarias:

$$F_i \sim (c_{i1}, \ldots, c_{in})$$

para $i = 1, \ldots, m$

3.- Multiplicación de funciones.

Consideremos dos funciones F_i y F_k con $i<k$ (de modo que F_k es de altitud no inferior a la de F_i) y los vectores de n componentes ternarias

$$F_i \sim (c_{i1}, \ldots, c_{ij}, \ldots, c_{in}),$$
$$F_k \sim (c_{k1}, \ldots, c_{kj}, \ldots, c_{kn}),$$

que les corresponden en virtud de la tabla de codificación. A partir de estas expresiones construiremos ahora un nuevo vector de n componentes cuaternarias que designaremos por $F_k.F_i$

$$F_k.F_i \sim ((c_{ik})_1, \ldots, (c_{ik})_j, \ldots, (c_{ik})_n)$$

y denominaremos producto de las funciones F_i y F_k; las componentes de este vector producto pueden tomar solamente uno de los cuatro valores 0, 1, 2 ó 3 y se determinan aplicando determinadas reglas que, para abreviar la exposición, hemos resumido esquemáticamente en la figura

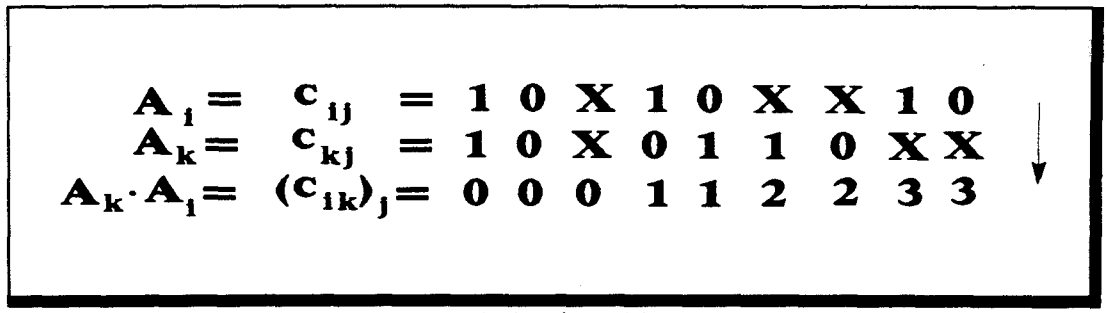


Figura 3.46 ***Reglas de Multiplicación***

4.- Tabla de Multiplicación y clasificación de los conjuntos de datos (o variables).

Esta tabla tiene (figura 3.47) dos entradas que corresponden ambas al conjunto ordenado de las acciones, y un elemento cualquiera m_{pq} de la misma tiene un valor asignado solamente si está definido el producto $F_p.F_q$, de modo que los elementos de la diagonal principal y los situados por encima de ésta no tienen ningún valor asignado; los elementos m_{pq} con $p>q$ toman solamente uno de los tres valores 0, 1 ó 2 según la estructura que tenga el producto respectivo $F_p.F_q$; concretamente,

m_{pq} = 0 si entre las componentes del producto $F_p.F_q$ existe al menos una igual a 1;

m_{pq} = 1 si todas las componentes del producto $F_p.F_q$ son iguales a 0 o bien a 2;

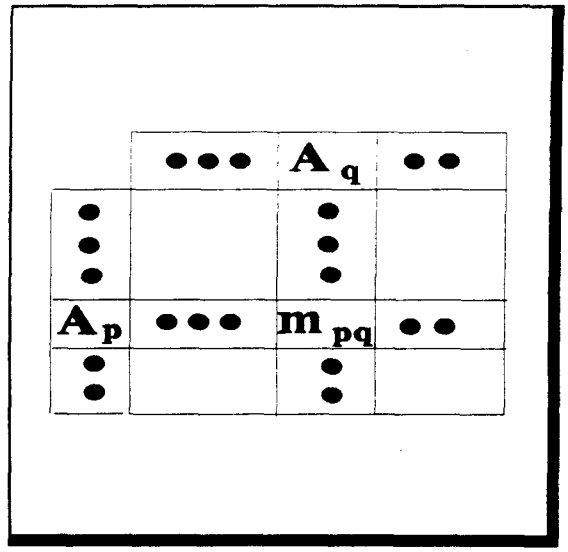


Figura 3.47 Tabla de Multiplicación

m_{pq} = 2 si las componentes del producto $F_p.F_q$ toman valores correspondientes a las combinaciones (0, 2, 3) o (2, 3) solamente.

En función de los resultados se dice que:

- las funciones de una Acción actúan sobre unas variables *disjuntas* si la tabla de multiplicación comprende elementos iguales a 0 solamente, y por tanto la solución óptima es la *arborescencia*.

- las funciones de una Acción actúan sobre unas variables *incluidas* si la tabla de multiplicación comprende elementos iguales a 1 solamente, y por

consiguiente la solución óptima también es la arborescencia.

- las funciones de una Acción actúan sobre unas variables *intersectadas sin inclusión* si la tabla de multiplicación comprende elementos iguales a 2 solamente, por tanto la solución óptima es la *alternativa compleja.*

En este mismo sentido diremos a veces que las funciones están condicionadas por variables disjuntas, incluidas o en intersección sin inclusión; y en este caso una solución óptima es una *combinación de arborescencia y alternativa compleja.* Para construir la tabla de multiplicación puede usarse el algoritmo siguiente.

Paso 1: Poner $p = 2$
Paso 2: Poner $q = 1$
Paso 3: Calcular las componentes de A_p. A_q
Paso 4: Calcular el elemento m_{pq}
Paso 5: Si $q \neq p$, poner $q = q+1$ e ir al Paso 3
Paso 6: Si $p \neq m$, poner $p = p+1$ e ir al Paso 2
Paso 7: Fin

Figura 3.48 ***Algoritmo para la construcción de la Tabla de Multiplicación***

3.2.10.- DIAGRAMA WARNIER-ORR

El diagrama de Warnier-Orr es una representación gráfica que permite mostrar la estructura lógica de un conjunto de información así como la de una proceso. En el dominio del análisis la emplearemos para representar en la base de información la estructura lógica de los flujos de datos tanto de entrada como de salida a un proceso. También la emplearemos para representar la estructura funcional de cada proceso.

A continuación se muestra un diagrama general de Warnier-Orr, así como el significado de los diferentes elementos que en él participan.

Significado de los elementos

1.- Nombre lógico de un conjunto.
2.- Nombre de un subconjunto del conjunto.
3.- Nombre de un elemento (atómico).
4.- Símbolo para el "or exclusivo". Los componentes separados por este símbolo son entre sí exclusivos.
5.- Indicación de repetición.
6.- Número de veces que un elemento ocurre.
7.- Símbolo de referencia para una condición.

8.- Símbolo para el "or no exclusivo". Los componentes separados por este símbolo son entre sí independientes.

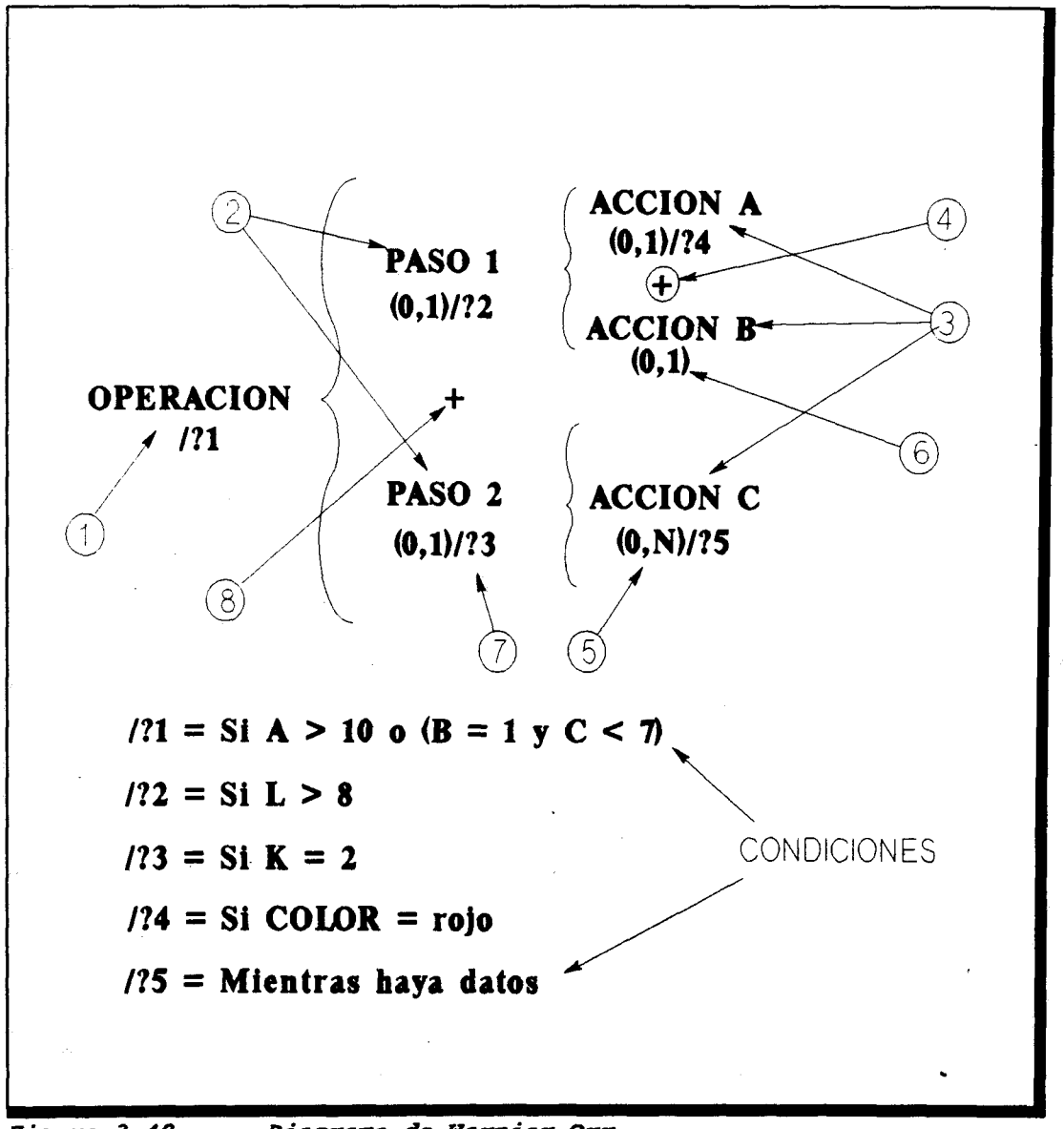


Figura 3.49 Diagrama de Warnier-Orr

3.3.- PROCEDIMIENTO DE DESARROLLO

3.3.1.- INTRODUCCION

La metodología propuesta tiene aplicabilidad en sistemas de información de ámbito general, es decir con características tanto de bases de datos como de tiempo real. Para ello hemos recogido los tres aspectos necesarios para su desarrollo. Cada aspecto va a constar de una determinada vista del sistema, siendo estas vistas las siguientes:

Vista de los Procesos

Esta es la vista más común de los sistemas de procesos de datos. La noción básica de esta vista consiste en que cada proceso cambia la información de entrada en información de salida. El sistema es visto estrictamente como un transformador de información, esta información la consideraremos siempre lógica y no física.

Esta vista queda representada por un modelo, que llamaremos en la metodología "***Modelo Esencial de Comportamiento***", en donde los requisitos funcionales del sistema a desarrollar y las interconexiones entre estas

funciones quedan reflejadas.

Vista de la Información

Esta vista se ha empleado para soportar el desarrollo de base de datos. La vista de información proveerá a la persona encargada de realizar el desarrollo software con medios para capturar los hechos que son vitales para el entendimiento y documentación de las características de lo que podemos denominar como datos.

Esta vista quedará representada por lo que denominamos en la metodología "***Modelo Conceptual de Datos***" y por los "***Flujos de Datos***" que fluyen en el Modelo Esencial de Comportamiento del sistema.

Vista de Eventos

La vista de los eventos es un aspecto fundamental para aquellos sistemas con características de tiempo real. Con esta vista podremos representar todos aquellos acontecimientos que ocurran fuera del sistema a desarrollar y a los cuales el sistema debe responder. A estos los denominamos "***eventos externos***".

La vista de eventos queda reflejada en las especificaciones de las transformaciones de control, así como en el modelo del entorno y el modelo esencial de comportamiento del sistema.

3.3.2.- FASES DE LA METODOLOGIA MIDES

El punto de partida para la metodología surge como consecuencia de una petición de una empresa para el desarrollo de un sistema de información automatizado.

Las entradas a la metodología son los objetivos generales marcados por la Dirección de la empresa y los objetivos específicos de cada Unidad Organizativa implicada en el desarrollo del futuro sistema. Las salidas de la metodología son:

- La Especificación de Requisitos Software del sistema.
- El Modelo Lógico de Datos
- El Modelo Lógico de Procesos.
- Estructura Lógica de las Operaciones.
- Especificación de las Acciones.

La información queda recogida en una Base de Información, entendiendo por ésta el lugar de almacenamiento de todos los productos software elaborados en el desarrollo.

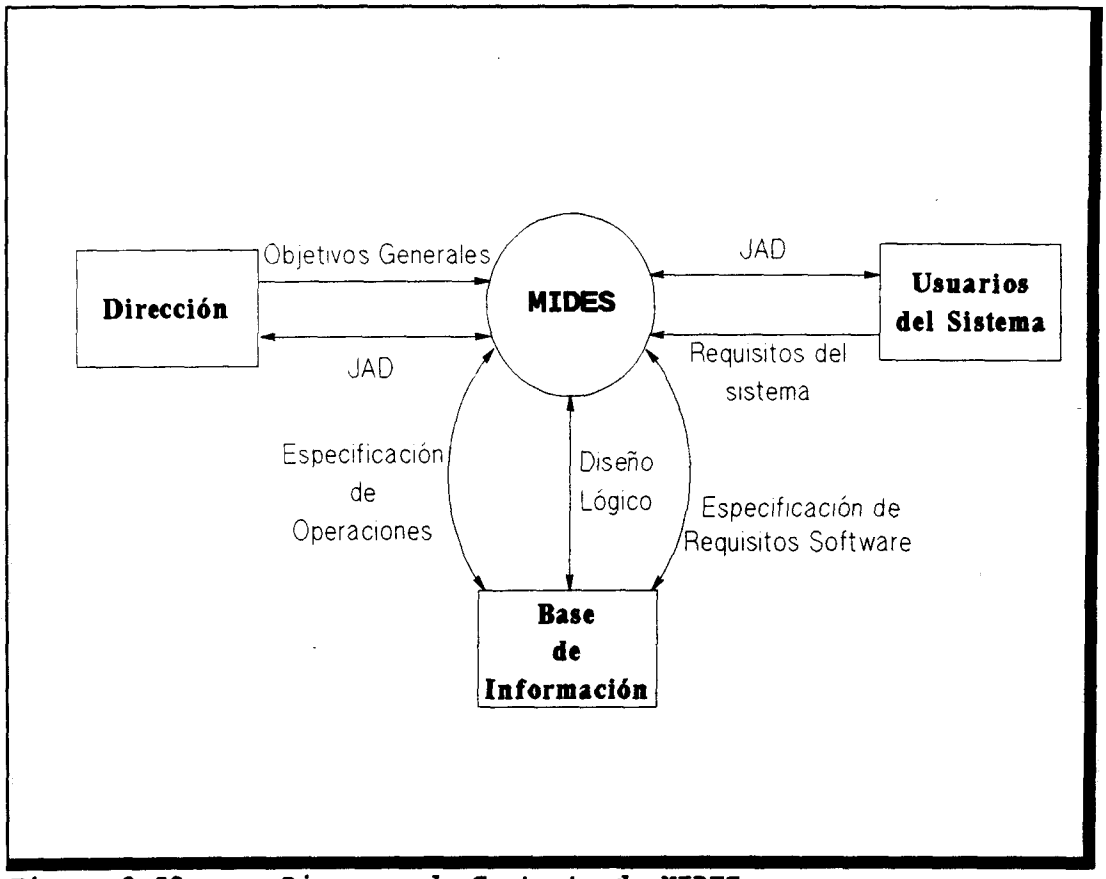


Figura 3.50 ***Diagrama de Contexto de MIDES***

La metodología consta de dos Fases, cada una con unos objetivos claros y un alcance limitado, teniendo un conjunto de salidas definido que proporciona una intercomunicación clara y limpia entre las fases. La figura 3.51 muestra las fases y etapas de la metodología.

La división en estas etapas proporcionará a la gestión del proyecto una gran ayuda, ya que la monitorización de la terminación de cada etapa es relativamente fácil debido a que cada etapa es autocontenida, tiene un objetivo preciso y produce un conjunto de salidas determinadas. Además al final de cada etapa se realizará un proceso de verificación y validación controlando de esta manera el progreso y la calidad de cada uno de los productos que se obtienen en el proceso de desarrollo software.

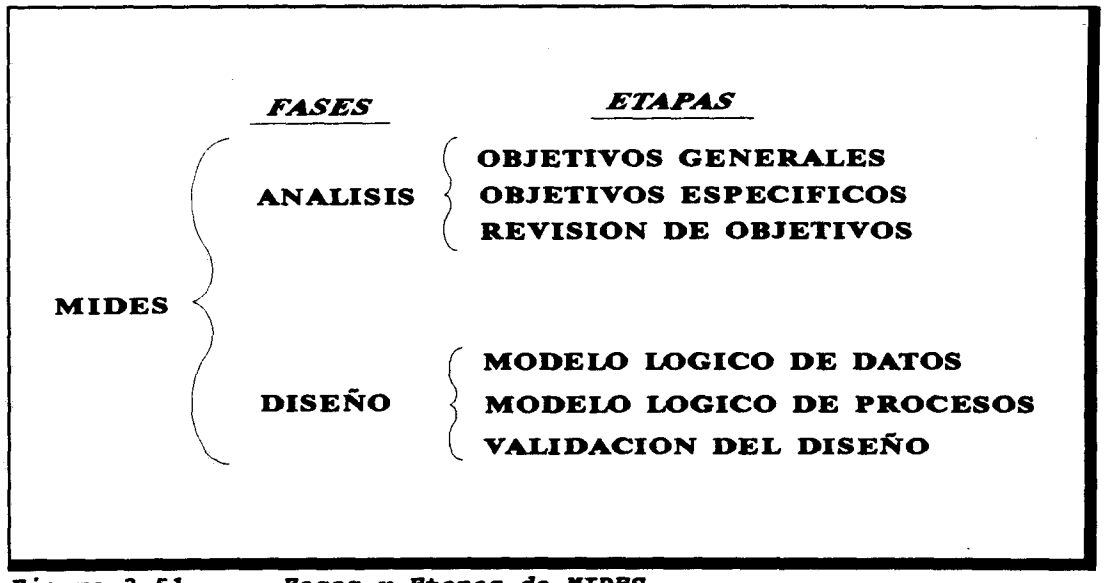


Figura 3.51 ***Fases y Etapas de MIDES***

3.3.3.- ETAPAS DE LA METODOLOGIA

3.3.3.1.- ESTABLECIMIENTO DE LOS OBJETIVOS GENERALES

Objetivo: Se trata de determinar el alcance del sistema a desarrollar.

Entrada: Se recibe una petición de mecanización de un sistema.

Salida:
- Diagrama de Objetivos
- Diagrama de Contexto
- Reglas de Gestión

Técnica: JAD. Proponemos que esta técnica se aplique en esta etapa únicamente al grupo directivo de la empresa por ser éstos los que poseen una visión clara de los objetivos de alto nivel a lograr.

Detalle: En esta etapa sucede el primer conocimiento del problema propuesto y por tanto se debe comprender la razón por la que se le requiere. Se replantean los problemas que se le someten en una vista de conjunto, y se obtiene como resultado de las entrevistas un Diagrama de Objetivos inicialmente y a partir de éste se elabora un Diagrama de Contexto. Durante el proceso de esta etapa

se recogen las Reglas de Gestión que se perciban. Debe de realizarse un proceso de validación una vez establecidos estos documentos para enseguida y definitivamente estabilizarlos. Por último se debe obtener de la dirección el grupo de personas de las áreas implicadas a participar en la siguiente etapa.

3.3.3.2.- ESTABLECIMIENTO DE OBJETIVOS ESPECIFICOS

Esta etapa la hemos descompuesto en dos actividades:

- Análisis del Sistema Actual
- Análisis del Sistema a diseñar

El objetivo último de esta etapa es la de especificar el sistema que se desea diseñar. Para ello inicialmente se recogen todas las operaciones efectuadas y todas las informaciones manipuladas sobre cada puesto de trabajo que pertenece al campo de estudio, para posteriormente pasar a describir todos los requisitos del sistema a diseñar. Será de gran importancia para el logro de tales objetivos:

- que los procesos a realizar en esta etapa sean llevados a cabo por analistas que reunan estas dos características: bastante experiencia práctica y

dominio de las técnicas que proponemos en esta etapa.

- una adecuada selección de los usuarios que van a participar en el desarrollo de la especificación del sistema. Estos deberán ser elegidos por la dirección.

ANALISIS DEL SISTEMA ACTUAL

Este proceso sólo lo realizaremos si los analistas de sistemas no están familiarizados con el área del sistema en la que estén trabajando, y en cualquier caso este análisis siempre será del sistema lógico; logrando con ello un ahorro considerable en el tiempo de desarrollo, ya que el 75% del análisis físico actual no nos sirve para el futuro desarrollo, y evitando así mismo el peligro potencial de la cancelación del proyecto por no conseguir un sistema que funcione en el tiempo planificado.

Las entradas a esta actividad son:

- el Diagrama de Objetivos
- el Diagrama de Contexto
- Reglas de Gestión

Las salidas producidas son:

- los Diagramas de Flujo de Datos lógicos del sistema actual
- una lista de problemas y de nuevos requisitos.

Los objetivos de este análisis son:

- Determinar y describir las tareas ejecutadas. En particular, se atribuirá para cada tarea a precisar los acontecimientos que la desencadenan, la periodicidad de ejecución y su duración, los datos manipulados y su volumen, y las reglas que se aplican para ejecutar el trabajo.
- Observar la circulación de las informaciones. Esta será observada entre los diferentes puestos de trabajo, principalmente a través del soporte de documentos escritos u otros medios tales como mensajería electrónica, fax, etc.
- Aprender el lenguaje de la empresa. Su buen conocimiento y buena comprensión será para el analista el elemento necesario para el establecimiento de un diálogo claro y preciso.

Las tareas a realizar en este análisis son:

Investigar sistema actual

Objetivo: Documentar las funciones, datos y controles en el sistema actual.

Entrada:
- Diagrama de Objetivos
- Diagrama de Contexto

Salida:
- Conjunto de notas de investigación
- Reglas de Gestión
- Reglas de Cálculo
- Reglas de Organización

Técnica: JAD.

Proceso: Proponemos que por parte de la empresa asistan a las reuniones de conjunto en esta etapa únicamente los usuarios pertenecientes al campo de estudio.

Crear Diagramas de Flujo de Datos lógicos del sistema actual

Objetivo: Realizar un modelo que represente de una manera lógica lo que el sistema actual realiza. En este modelo están incluidas las funciones lógicas, los flujos de datos, almacenes de datos y entidades externas asociadas con el sistema actual.

Entrada: Notas de investigación del sistema actual y reglas de gestión, cálculo y de organización

Salida:
- Diagramas de Flujo de Datos de la lógica actual
- Lista de problemas y de nuevos requisitos

Técnica: Diagramas de Flujo de Datos

Detalle: El modelo lógico representa lo que el sistema actual realiza, y no cómo opera. La realización del modelo debe ser realizada conjuntamente usuario con analista, teniendo el analista la responsabilidad de obtener un producto técnicamente correcto y el usuario la de validar y aprobar el modelo obtenido en esta tarea.

Debe elaborarse, a la vez que se va especificando el sistema actual, una lista de los problemas encontrados y las peticiones para nuevas funciones. Proponemos que ambos se documenten formalmente en esta tarea según se vayan revelando en el proceso de construcción de los DFD.

ANALISIS DEL SISTEMA A DISEÑAR

Esta actividad tiene por objetivo acordar con el usuario lo que el sistema deberá hacer, y recoger y estructurar toda la información necesitada para el funcionamiento del sistema, independientemente de las funciones y aplicaciones que las usen.

Las entradas para llevar a cabo este análisis son los Diagramas de Flujo de Datos lógicos del sistema actual y la lista de problemas y de nuevos requisitos. Las salidas producidas son un Modelo Conceptual de Datos preliminar y un ***Modelo Esencial*** Preliminar del sistema a desarrollar. El Modelo Esencial consta de dos componentes principales:

- ***Modelo del Entorno***: define la frontera entre el sistema y el resto del mundo. Está compuesto por el Diagrama de Contexto y una Lista de Eventos.
- ***Modelo de Comportamiento***: describe el comportamiento interno del sistema requerido para interactuar adecuadamente con el Entorno.

Las tareas a realizar son:

Crear lista de eventos

Pasamos a detallar inicialmente en qué consiste esta lista.

La lista de eventos es una lista narrativa de "estímulos" que ocurren en el exterior y al cual nuestro sistema debe responder. Ejemplos de un sistema de pedidos de libros:

- *Un cliente realiza un pedido. (F)*
- *Un cliente cancela un pedido. (F)*
- *La dirección requiere un informe de ventas. (T)*
- *Una orden de reimprimir un libro. (C)*

Cada evento está etiquetado con una F, una T o una C. Esto indica si es un evento orientado a flujos, un evento temporal, o un evento de control. Un ***evento orientado a flujos*** es aquel que está asociado con un flujo de datos; es decir, que el sistema percibe que el evento ha ocurrido cuando algún dato, o posiblemente un conjunto de datos, ha entrado al sistema. En un Diagrama de Contexto no todos los flujos de datos son eventos, puede suceder que el sistema explícitamente solicite entradas a otras Entidades Externas para producir alguna respuesta al sistema.

Por esto no hay una correspondencia de uno a uno en el diagrama de contexto y la lista de eventos. En general, cada flujo de datos es o bien un evento (o más preciso, la indicación de que el evento ha ocurrido), o bien es información requerida por el sistema para procesar un evento.

Además de los eventos orientados a flujos, un sistema puede tener también eventos temporales. Como el nombre indica, los ***eventos temporales*** son disparados en un momento determinado en el tiempo. Ejemplos de eventos temporales son:

- *Se requiere un informe diario de todos los pedidos de libros a las nueve de la mañana.*
- *Las facturas se deben generar a las tres de la tarde.*
- *Se deben generar en cada hora los informes a la dirección.*

Los eventos temporales no se disparan con la entrada de datos; nos podemos imaginar que el sistema tiene un reloj interno el cual determina el paso del tiempo. Un evento temporal puede requerir que el sistema solicite entradas de alguna Entidad Externa.

Los *eventos de control* se pueden considerar como un caso especial de un evento temporal: un estímulo externo que sucede en un punto imprevisible en el tiempo, por tanto el sistema no puede anticiparlo mediante el uso de un reloj interno (como sucede con los eventos temporales), ni por la presencia de la llegada de datos (como sucede con los eventos orientados a flujos). Un evento de control está asociado con un flujo de control en el Diagrama de Contexto.

Un flujo de control puede ser considerado como un flujo de datos binario, puede conmutar de un estado a otro en cualquier momento, y por tanto señala al sistema que debe tomar alguna acción inmediata. Los sistemas de información generalmente no tienen flujos de control en sus Diagramas de Contexto, sin embargo en los sistemas de tiempo real son bastante frecuentes.

Entrada: - Diagrama de Contexto
- Diagramas de Flujo de Datos lógicos del sistema actual
- Lista de problemas y de nuevos requisitos

Salida: Lista de Eventos

Técnica: JAD

Procedimiento de construcción:

La lista de eventos es una lista textual simple de los eventos en el entorno al cual el sistema debe responder. Cuando construyamos la lista de eventos debemos asegurarnos de distinguir entre un evento y un flujo relativo a evento. Por ejemplo, esto probablemente no sea un evento: "*El pedido de cliente es recibido por el sistema*". En cambio, en este otro ejemplo: "*El cliente realiza un pedido*", probablemente sea el flujo de datos por el cual el sistema detecta que el evento ha ocurrido.

Si describimos el evento desde el punto de vista del sistema, es decir desde dentro del sistema mirando hacia afuera, podremos equivocarnos en la identificación de flujos de entrada que no son eventos por sí mismos pero que son requeridos para procesar algún otro evento. Por ello es mejor describir los eventos desde el punto de vista del entorno, es decir desde fuera del sistema mirando hacia el interior del mismo.

En la mayoría de los casos, la manera más fácil de identificar los eventos relevantes para un sistema es visualizar el sistema en acción. Para ello examinamos cada Entidad Externa y nos preguntamos qué efecto pueden tener las acciones de la Entidad Externa en el sistema. Esto debe ser realizado junto con el usuario del sistema, el cual

puede jugar el papel de Entidad Externa. Sin embargo hay que examinar con detenimiento los eventos discretos pues pueden estar constituidos por dos eventos simples. Esto sucede bastante con eventos orientados a flujos. Debemos examinar el evento candidato y preguntarnos si todas las ocurrencias del evento llevan los mismos datos; si los datos están presentes en algunos casos y en otros no, podemos tener realmente dos eventos distintos. Por ejemplo, si nos detenemos en el evento "*Un cliente realiza un pedido*", podemos averiguar que en algunas ocurrencias del evento está incluido el elemento "identificador del vendedor" y en otras ocurrencias no; y podemos averiguar que la respuesta al sistema es diferente si un vendedor está implicado o no lo está. Por eso sería apropiado tener dos eventos separados: "*Cliente realiza pedido*", y "*Vendedor realiza pedido del cliente*".

En la lista de eventos se deben incluir no sólo interacciones normales entre el sistema y sus Entidades Externas, sino también situaciones de fallo. Como estamos creando un Modelo Esencial, es decir con una tecnología perfecta, no debemos preocuparnos de fallos de nuestro sistema, pero sí de tener en cuenta los fallos o errores posibles de las Entidades Externas. Por ejemplo, el evento "*Llega al almacén orden de reimprimir el libro*". Pero ¿qué

ocurre si no llega en el plazo esperado por el editor? ¿qué hará el sistema?. Probablemente necesitaremos un evento adicional en el sistema.

Verificar Modelo del Entorno

Objetivo: Consiste en verificar la consistencia de los dos componentes del Modelo del Entorno

Entrada: - Diagrama de Contexto
- Lista de Eventos

Salida: Modelo del Entorno verificado

Técnica: Inspección

Procedimiento:

- cada flujo de entrada en el Diagrama de Contexto deberá ser necesario para que el sistema reconozca que ha ocurrido un evento, o que es necesario por el sistema para producir una respuesta a un evento, o ambos.

- cada flujo de salida deberá ser una respuesta a un evento.

- cada evento no temporal en la lista de eventos deberá tener entrada al sistema para que el sistema pueda detectar que ha ocurrido un evento.

- cada evento deberá producir una salida inmediata como respuesta, o deberá almacenar datos para ser salida más tarde (como respuesta o parte de una respuesta a algún otro evento), o causará que el sistema cambie su estado (como se indica en un Diagrama de Transición de Estados).

Crear Modelo Esencial Preliminar

Objetivo: Desarrollar un modelo de comportamiento del sistema que el usuario desea. En este modelo se asume que disponemos de una tecnología perfecta y que su coste es nulo.

Entrada:
- Diagramas de Flujo de Datos del sistema lógico actual
- Lista de problemas y nuevos requisitos
- Modelo del Entorno

Salida: Modelo Esencial Preliminar

Técnica: Diagramas de Flujo de Datos.

Procedimiento:

La aproximación que proponemos no es una aproximación de arriba hacia abajo ni de abajo hacia arriba, es una aproximación del centro hacia afuera. Después de que se hayan desarrollado los DFD iniciales puede ser que se requiera un agrupamiento de procesos para

crear un proceso de nivel superior, o puede ser que sea necesaria alguna descomposición adicional de las funciones obtenidas.

La siguiente aproximación implica los siguientes pasos:

1.- Se dibuja un proceso por cada evento en la lista de eventos.
2.- Al proceso se le da el nombre que describa la respuesta que el sistema debería dar al evento asociado.
3.- Se dibujan las entradas y las salidas de tal manera que el proceso sea capaz de realizar la respuesta requerida. Los almacenes se dibujan cuando sean apropiados para la comunicación entre procesos.
4.- El DFD resultante se valida con el Diagrama de Contexto y la Lista de Eventos para comprobar si está completo y si es consistente.

El **primer paso** es hacia adelante, además casi de una manera mecánica. Si hay 25 eventos en la lista de eventos, deberemos dibujar 25 círculos. Deberemos numerar el proceso de tal manera que coincida con el evento asociado. Por eso

el evento 13 corresponde con el círculo (proceso) 13. Más tarde renumeraremos los procesos.

El **segundo paso** es también hacia adelante y mecánico: a cada proceso se le da un nombre apropiado basado en la respuesta que tiene que obtener. Para ello deberemos examinar el evento y preguntarnos a nosotros mismos qué respuesta se supone que debe dar el sistema a este evento. Por ejemplo si un evento es EL CLIENTE REALIZA UN PAGO, un nombre adecuado de proceso podría ser ACTUALIZAR LAS CUENTAS RECIBIDAS (si es la única respuesta que se requiere del sistema), en vez de PROCESO DE PAGO DE CLIENTE (el cual no nos dice nada de la naturaleza de la respuesta).

El **tercer paso** no es mecánico, pero sí es hacia adelante. Para cada proceso dibujado, debemos identificar las entradas que el proceso necesita para llevar a cabo su trabajo; debemos identificar las salidas que el proceso produce y los almacenes a los que el proceso debe acceder. Esto se realiza normalmente mediante entrevistas al usuario (si la información no está reflejada en los DFD del sistema actual) y concentrándonos en cada evento y su proceso asociado. Deberemos preguntar al usuario:

- ¿qué necesita este proceso para realizar su trabajo?
- ¿qué salidas genera?

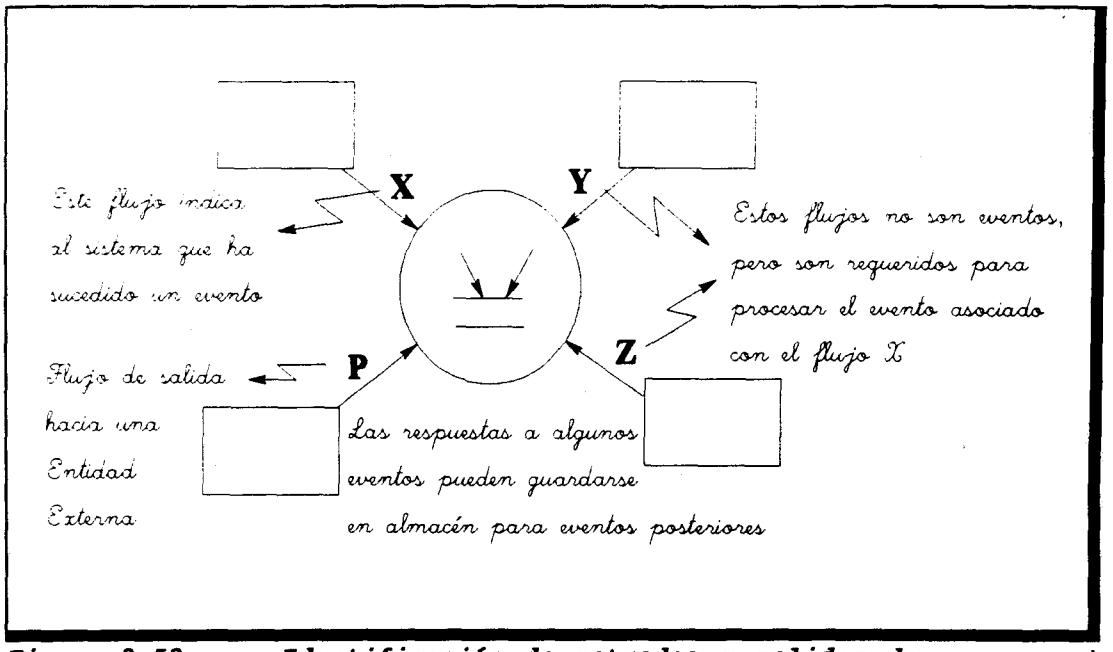


Figura 3.52 ***Identificación de entradas y salidas de un proceso preliminar***

En muchos casos el evento está orientado a flujos; esto es, el sistema reconoce la ocurrencia del evento por la llegada de datos del exterior. Esto significa que el flujo de datos asociado debe estar conectado al proceso requerido para responder a ese evento, pero puede ser que se necesiten entradas adicionales (de otras Entidades Externas y posiblemente de almacenes) para producir la salida requerida. De igual manera debemos dibujar las salidas producidas por los procesos como parte de la respuesta. En muchos casos, esto implicará salidas hacia el exterior del sistema, pero también puede implicar salidas que son enviadas a almacenes para que sean usadas

como entradas para otros procesos.

Finalmente, **el cuarto paso** es una actividad de comprobación similar a la aplicación de las reglas de balanceo. Debemos verificar que cada entrada que aparece en el Diagrama de Contexto está asociada con una entrada en uno de los procesos del DFD preliminar, y que además cada salida producida por un proceso en el DFD preliminar está también en el Diagrama de Contexto.

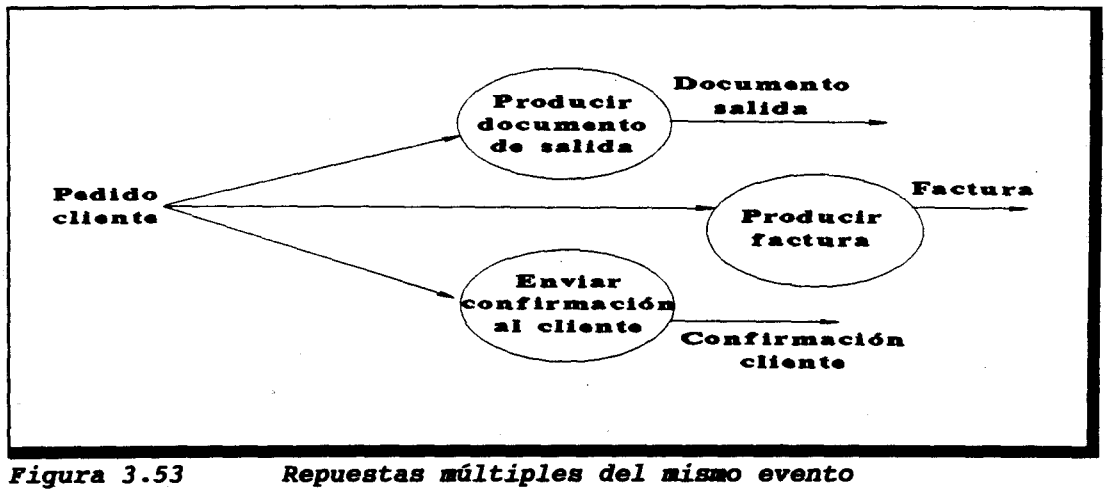


Figura 3.53 ***Repuestas múltiples del mismo evento***

Hay dos casos especiales:

1.- Un único evento que causa múltiples repuestas. En este caso cada evento entra en un proceso distinto si las salidas que va a producir son independientes una de las otras.

2.- Múltiples eventos que causan la misma respuesta. Habrá situaciones donde un proceso está asociado con más de un evento. Esto será válido cuando la respuesta realizada por el proceso sea idéntica para los distintos eventos y sólo cuando los datos de entrada y salida sean idénticos para las distintas respuestas de eventos.

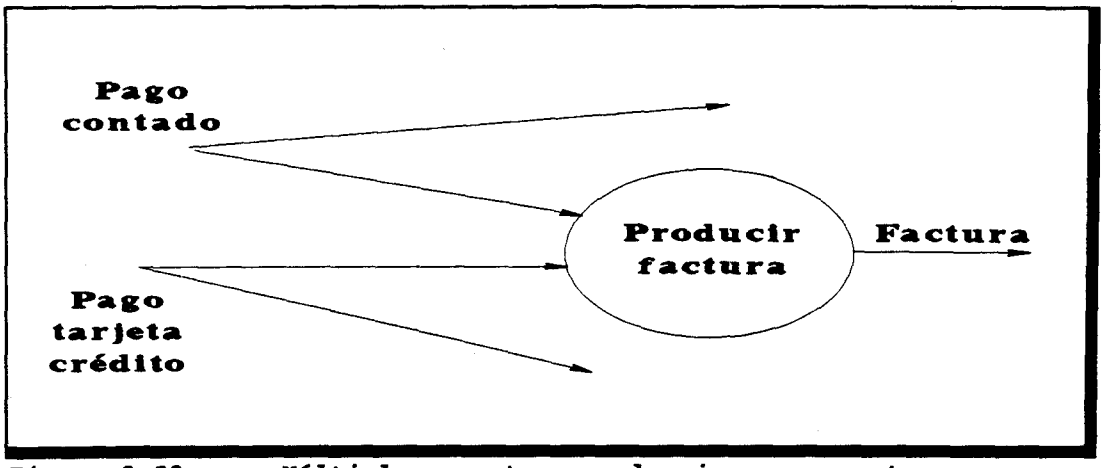


Figura 3.53 ***Múltiples eventos con la misma respuesta***

Los procesos obtenidos en el DFD preliminar no están conectados unos con otros ni directamente ni a través de almacenes. Esto es porque los procesos que han surgido en el DFD preliminar representan respuestas a un evento, y además los eventos que ocurren en el entorno exterior son en general asíncronos. La manera de sincronizar múltiples eventos interdependientes es a través de un almacén. Este almacén será un almacén esencial (lógico); es decir un

almacén que se requiere no por demoras de tiempo asociadas con tecnología imperfecta, sino por consideraciones de tiempo en el entorno.

Crear Modelo Esencial

Objetivo: Obtener el modelo de comportamiento del sistema deseado para presentación y validación de los usuarios.

Entrada: Modelo Esencial Preliminar

Salida: Modelo Esencial

Técnica:
- Diagramas de Flujo de Datos
- Walkthrough

Procedimiento:

El Modelo Esencial Preliminar no está preparado para que sea presentado y validado por los usuarios porque es muy complicado, ya que constará de un único nivel con tantos procesos como eventos se hayan detectado, y por existir una ausencia de aplicación de los principios de orden jerárquico y de abstracción que hagan legible el Modelo. Los pasos a realizar son los siguientes:

Construcción de diagramas padres

1.- Agrupar procesos que estén relacionados en un

único proceso con significación, cada uno de estos representará un proceso en el diagrama de nivel superior. Procedimiento para su realización:

- Cada grupo de procesos debe de estar relacionado en sus respuestas; generalmente los procesos están trabajando casi con los mismos datos.
- Aplicar el principio de información oculta en lo que respecta a almacenes locales. Si observamos un grupo de almacenes en el DFD preliminar que están interactuando con un almacén, y ningún otro proceso del DFD preliminar se relaciona con ese almacén, entonces podemos crear un proceso de nivel superior que oculte el almacén.
- Mantener una especificación legible; para ello aconsejamos que un DFD no conste de más de 7 más o menos de 2 componentes de información, donde cada componente es un proceso con sus flujos de información o bien un almacén.

Construcción de diagramas hijo

Puede ser que necesitemos expandir un proceso del DFD preliminar. Esto ocurrirá cuando identifiquemos que un proceso en DFD preliminar no es una función primitiva (su descripción es compleja) y por tanto requiere una descomposición del mismo. Procedimiento para la creación de diagramas hijos:

Proceso que está realizando una función compleja

Intentar identificar las subfunciones. Cada una de las cuales se debe poder llevar a cabo en un nivel inferior. Por ejemplo, supongamos que tuviéramos un proceso llamado "*Ajustar trayectoria del misil*"; este podría ser el proceso responsable del manejo de un evento temporal en un sistema de guía de misiles. La función general de ajustar la trayectoria podría ser descompuesta en las siguientes subfunciones:

- Calcular variación coordenada X
- Calcular variación coordenada Y
- Calcular variación coordenada Z
- Calcular nuevo factor atmosférico
- Calcular nueva velocidad del viento
- etc.

Número de flujos de entrada y de salida al proceso

Si el número de flujos de entrada y de salida a un proceso es de una relación de "n a m", se realizará la descomposición hasta que la relación sea de "1 a 1" o bien de "n a 1".

3.3.3.3.-REVISION DE OBJETIVOS

Objetivo: Comprobar que es correcta técnicamente la especificación y si está de acuerdo con los requisitos del usuario.

Entrada: Modelo Esencial

Salida: Modelo Esencial Verificado y Validado

Técnica: Walkthrough

Procedimiento:

Las personas que vayan a participar en la revisión deberán comprobar las siguientes propiedades en orden secuencial:

Completo

- Comprobar que todos los elementos del DFD están numerados e identificados al igual que los diagramas.

- Comprobar que se han definido todos los componentes del DFD en la Base de Información.
- Comprobar que todos los procesos primitivos tienen una especificación de proceso.
- Comprobar que todos los procesos de control tienen asociado un diagrama de transición de estados.

Integridad

Consiste en asegurarse de la integridad entre los componentes del Modelo Esencial. Para ello deberemos fijarnos en los siguientes problemas:

- Todas las referencias de datos que son realizadas por una especificación de proceso deben ser entradas en la Base de Información.
- Todas las entradas a un círculo y las salidas producidas por él deberán corresponder a las entradas y salidas descritas en la especificación del proceso.
- Errores de balanceo.
- Procesos o almacenes de datos en los que sólo se producen entradas pero no hay salidas.
- Las salidas de un círculo son derivables de las entradas al mismo.
- Cada proceso en un DFD deberá recibir únicamente las

entradas esenciales para realizar sus tareas.

Estas dos propiedades (completo e integro) son una tarea mecánica que por lo tanto pueden ser realizadas por una herramienta. Con esto queda verificado el producto que hemos realizado pero hemos de validarlo, para ello debemos comprobar las siguientes propiedades:

Exactitud

Lo que se pretende comprobar en este momento es si realmente hemos descrito los requisitos del usuario para el nuevo sistema. Esta es una razón para incluir en este momento a uno o más usuarios en el Walkthrough. Deberemos comprobar:

- que los requisitos del sistema se describen de una manera en la que se ignoren las restricciones tecnológicas
- que no se han descrito las necesidades de un usuario particular o de un departamento determinado
- las restricciones de implantación que el usuario impone para que el sistema sea aceptable
- otras restricciones como la del tiempo de respuesta, entorno, formato físico de algunas de las entradas o salidas
- que las restricciones impuestas por el usuario sobre

la implantación del sistema son razonables

Calidad

Aunque el Modelo Esencial sea completo, consistente internamente, y refleje exactamente los requisitos del usuario, puede no ser aún un producto adecuado. El equipo de la revisión se deberá cuestionar el estilo, la organización y la facilidad de mantenimiento de los documentos de análisis.

Para ello proponemos que el Modelo Esencial, una vez validado, se documente de acuerdo con alguno de los prototipos propuestos en [IEE84]. Esta labor también puede ser realizada por una herramienta.

3.3.3.4.- DISEÑO DEL MODELO LOGICO DE LOS DATOS

El objetivo de toda concepción de un Sistema de Información es establecer una base de datos que recoja todos los datos de la organización y memorizarlos sobre una estructura física, también conocida como Modelo Interno, según la terminología del grupo ANSI/SPARC.

Objetivo: Crear un modelo intermedio único que sirva

de unión por un lado a los responsables de la organización y a los informáticos, y por otro que nos proporcione la independencia precisa entre los distintos entornos de la organización.

Entrada: - Diagrama de Objetivos
- Modelo Esencial, que refleja la realidad percibida por la organización

Salida: Modelo Lógico de Datos

Técnica: Formalismo Individual

Procedimiento:

Esta etapa consta de las siguientes tareas:

Creación de un Modelo Lógico de Datos Bruto

Mediante un proceso de aproximaciones sucesivas construimos el modelo lógico de datos. El primer paso consiste en identificar los principales individuos-tipo a partir del Diagrama de Objetivos. En el siguiente obtenemos las relaciones-tipo e identificamos nuevos individuos-tipo a partir de las reglas de gestión, estando éstas recogidas en la Base de Información durante la fase de análisis. Por último se añaden las cardinalidades al modelo, éstas estarán determinadas por las reglas de gestión y de organización. Un ejemplo de este proceso está expuesto en

el apartado 3.2.3.

Depuración de los datos

Objetivo: Eliminar las redundancias, ambigüedades y contradicciones de los datos recogidos en la Base de Información.

Entrada: Todos los datos elementales de la Base de Información.

Salida: Datos depurados.

Técnica: Diseño asistido por ordenador.

Procedimiento:

Consiste en la realización de los siguientes pasos:

Asignación de palabras descriptivas

Para cada dato de la Base de Información se definen cinco "palabras descriptivas", que contribuyan a dotarlo de significado.

Eliminación de Polisemias

Se entiende por Polisemias aquellas palabras que tienen el mismo nombre y distintos significados. Su identificación es muy simple: basta con comparar los distintos datos y sus respectivas palabras descriptivas. Si se trata de la misma información (las cinco palabras descriptivas iguales), se elimina uno de los dos datos. Si

son dos informaciones diferentes el diseñador deberá cambiar el nombre de uno de ellos, que seguirá conservando todas sus palabras descriptivas.

Eliminación de Sinónimos

Se entiende por Sinónimos a aquellas palabras que tienen distintos nombres y el mismo significado. La búsqueda de sinónimos se realiza mediante el examen de las palabras descriptivas. Cuando dos datos tienen cuatro (o el número que considere el diseñador) palabras descriptivas idénticas son sinónimos. Una vez encontrados los sinónimos se borrará uno de ellos. Si el diseñador no los considera sinónimos deberá modificar sus palabras descriptivas.

Crear Modelo Lógico de Datos

Objetivo: Asignación de los datos a los individuos-tipo y relaciones-tipo.

Entrada: Datos depurados

Salida: Modelo Lógico de Datos

Técnica: Diseño asistido por ordenador conjuntamente con reglas inherentes al formalismo individual

Procedimiento:

El procedimiento a seguir consta de dos tareas:

- el diseñador examina cada dato decidiendo si dicho dato pertenece a un individuo-tipo o a una relación-tipo. Puede ser que en este procedimiento surja algún nuevo individuo o relación. Si surge una relación nos debemos preguntar con qué individuos está relacionada y con qué cardinalidades.
- determinar cuál de las propiedades del individuo puede ser el identificador. Si ninguna de ellas se considera apta se puede crear una propiedad ficticia que pueda asumir el papel de identificador, en tal caso deberá ser dada de alta en la Base de Información.

3.3.3.5.- DISEÑO LOGICO DE LOS PROCESOS

El objetivo de esta etapa es la de obtener las unidades de realización de que está compuesto el sistema, así como la estructura lógica de las mismas. Las entradas en esta etapa son el conjunto de acciones recogidas en la Base de Información durante la actividad de Análisis del sistema a diseñar. La salida que obtendremos será la determinación de los procesos de que está compuesto el sistema. Se entiende por ***Proceso*** a aquel conjunto de operaciones interrelacionadas. Las actividades a realizar

son las siguientes:

Agrupamiento de acciones en operaciones

Objetivo: Agrupar las acciones en operaciones.

Entrada: Las acciones recogidas en la Base de Información

Salida: Operaciones. Se entiende por ***Operación*** a una acción o conjunto de acciones que se desencadenan ininterrumpidamente en reacción a uno o varios eventos.

Técnica: Matriz de Procesos

Procedimiento:

- Poner las funciones obtenidas en el Modelo de Comportamiento en las columnas de la matriz.
- Poner los eventos obtenidos en la Lista de Eventos en las filas de la matriz.
- Los componentes de la matriz serán:
  - cero, si el evento no participa en la acción
  - uno, si el evento participa como entrada
  - menos uno, si el evento participa como salida

Aquellas filas que sólo tengan unos y ceros constituyen las entradas al Modelo Lógico de Procesos; aquellas que sólo tengan menos uno constituyen las salidas

al Modelo Lógico de Procesos; y las que tengan unos y menos uno son eventos intermedios.

- Agrupar las funciones que requieran de los mismos eventos de entrada, o que estén intercomunicados. Dos ejemplos se muestran en el apartado 3.2.5.

Determinación de procesos

Objetivo: Obtener los procesos del sistema.

Entrada: - Operaciones
- Eventos de entrada y salida de cada operación

Salida: Procesos

Técnica: Matriz de operaciones/eventos

Procedimiento:

Las filas de la matriz las constituyen las operaciones obtenidas en la actividad anterior, y las columnas están formadas por los eventos de entrada y salida de cada operación. Los componentes de la matriz tendrán una X si el evento participa tanto de la entrada como de la salida. Se unen las cruces que se encuentran en la misma fila y columna, y aquellas que hayan quedado unidas constituyen los procesos del sistema. Se muestra un ejemplo en el apartado 3.2.6.

Estructura Lógica de Procesos

Objetivo: Establecer la secuencia de ejecución de las operaciones de un proceso y verificarlas.

Entrada: - Operaciones de cada proceso
- Eventos del proceso

Salida: Estructura del proceso

Técnica: - Matriz de Precedencias
- Formalismo de Petri

Procedimiento:

Las filas de la matriz las constituyen las operaciones del proceso y las columnas los eventos del proceso. En las primeras filas estarán los eventos externos, colocando una cruz en las operaciones que participe. A continuación los eventos internos, colocando un 1 en la operación que lo produce y un -1 en la que lo requiere. Además se establece un vector de estado con tantos componentes como eventos internos inicializado a ceros, el cual nos indica los eventos disponibles, en cuyo caso éstos se encuentran a uno. El algoritmo de proceso con ejemplo de demostración está descrito en el apartado 3.2.7. El resultado de este proceso se describe gráficamente usando las redes de Petri. Proponemos esta representación por poseer unos procedimientos de verificación muy formalizados. Estos se hayan descritos detalladamente en [CUE89].

Estructura Lógica de las Operaciones

Objetivo: Diseñar la estructura lógica de cada operación y especificar las acciones internas a ella.

Entrada:
- Estructura Lógica del Proceso
- Modelo de Comportamiento

Salida:
- Estructura Lógica de Operación
- Tabla de Multiplicación

Técnica: Diagrama de Warnier-Orr

Procedimiento:

El procedimiento, algoritmo y ejemplos se halla detallado en [VEG85] y [CUE87].

3.3.3.6.- VALIDACION DEL DISEÑO DE DATOS

El objetivo de esta etapa es la de validar el diseño realizado de los datos; es decir, si están de acuerdo con lo establecido en la Especificación del Sistema. Como entrada tenemos, por una parte el diseño lógico de los datos y por otra el Modelo de Comportamiento. Como salida obtendremos el diseño lógico de datos validado. Las tareas a realizar son las siguientes:

Crear Modelos Externos

Objetivo: Diseñar la estructura de los datos de las modificaciones o consultas a la base de datos.

Entrada: Flujos de datos de entrada y de salida a los almacenes en el Modelo de Comportamiento.

Salida: Modelos Externos. Un ***Modelo Externo*** es una utilización particular de la base de datos. Puede tener por objeto poner al día unas ocurrencias (añadir, suprimir, modificar) en la base de datos o simplemente consultar informaciones almacenadas en la base.

Técnica: Formalismo Individual

Procedimiento:

- Obtener todas las referencias a los almacenes en el Modelo de Comportamiento;esto es, todos los flujos de entrada y de salida de cada almacén.
- Diseñar el Modelo Externo de cada referencia utilizando el formalismo individual.

Validar Modelo Lógico de Datos

Objetivo: Verificar que los Modelos Externos son deducibles del Modelo Lógico de Datos.

Entrada: Modelos Externos y Modelo Lógico de Datos.

Salida: Modelo Lógico de Datos validado.

Técnica: Matriz de Navegación.

Procedimiento:

Las dos funciones de actualización y consultas entrañan las siguientes reglas de validación. Utilizamos la técnica de Navegación (apartado 3.2.8) para la comprobación de estas reglas.

- Validación de propiedades externas.

  Una propiedad externa debe ser idéntica a una propiedad conceptual o ser deducible a partir de ésta.

- Validación de individuos externos.

  Una ocurrencia de un individuo externo debe ser expresable sin ambigüedad a partir de ocurrencias de individuos o de relaciones conceptuales.

- Validación de relaciones externas.

  Una relación externa es válida si los individuos externos de su colección son válidos.

- Validar las cardinalidades externas.

  Las cardinalidades deben estar incluidas en las cardinalidades conceptuales. Para esta regla no nos es útil la matriz de navegación.

Normalizar el Modelo de Datos

Objetivo: Evitar ambigüedades y redundancias que harían más difícil la tarea de conservar la integridad de los datos.

Entrada: Modelo Lógico de Datos validado

Salida: Modelo Lógico de Datos Normalizado

Técnica: Formalismos de la Normalización

Procedimiento:

No entramos en exponer el procedimiento porque es ampliamente conocido y descrito.

Resumen del procedimiento de desarrollo.-

La metodología MIDES consta de las fases de análisis y diseño, cada una de las cuales se desarrolla en tres etapas. Cada etapa se completa llevando a cabo una serie de tareas, éstas las hemos agrupado en actividades para su mejor comprensión. La figura 3.55 muestra un esquema de las tareas a realizar en la metodología MIDES.

MIDES

FASE	ETAPA	ACTIVIDAD	TAREA
Análisis	Establecimiento de los objetivos generales	Entrevistar directivos	Crear Diagrama de Objetivos Crear Diagrama de Contexto
	Establecimiento de los objetivos específicos	Análisis del sistema actual	Investigar sistema actual Crear DFD lógico actual
		Análisis del sistema a diseñar	Crear lista de eventos Verificar Modelo del Entorno Crear Modelo Esencial Prelim. Crear Modelo Esencial
	Revisión de objetivos	Verificación	Comprobar Completo-Integro
		Validación	Comprobar Exacto-Calidad
Diseño	Diseño Lógico de los Procesos	Crear Operac.	Matriz de Acciones
		Crear Procesos	Matriz Operaciones
		Especificación Procesos	Matriz Precedencias Formalismo de Petri
		Espec. Operación/ Acción	Tabla de Multiplicación Diagramas Warnier-Orr
	Diseño Lógico de los Datos	Crear Modelo de Datos Bruto	Crear Indiv/Relaciones Añadir Cardinalidades
		Depurar los datos	Asignar palabras clave Eliminar Polis./Sinonim.
		Crear Modelo Lógico Datos	Asignación propiedades
	Validación del Modelo Lógico de Datos	Validar Modelo	Crear Modelos Externos Matriz de Navegación
		Normalizar	Formalismo de Normalización

Figura 3.55 Tareas a realizar en MIDES

FUTURAS LINEAS DE INVESTIGACION

CAPITULO 4

4.- FUTURAS LINEAS DE INVESTIGACION

El trabajo cubre el ámbito de los requerimientos, análisis y diseño de una manera integrada. Un próximo trabajo podría ser la ampliación de la metodología CASE con la incorporación de la fase de construcción del sistema software. En esta fase de la construcción del sistema será interesante ahondar en los aspectos de la reusabilidad del código, con el fin de establecer componentes software lo suficientemente abstractos, generales y formales que permitan su empleo en diferentes sistemas, evitando así la redundancia del código en una instalación y generando como consecuencia el incremento de productividad.

Para ello sería necesario investigar en la generación de bibliotecas que recogieran en elementos reusables la base de información y conocimientos de cada empresa, así como en la generación de lenguajes de consulta y actualización de dicha biblioteca que permitan tanto incorporar nuevas informaciones como acceder a los componentes reusables adecuados solicitados para satisfacer unas especificaciones, señalando incluso el nivel de adecuación de los diferentes componentes proporcionados por medio del lenguaje.

Otra posible linea de investigación podrá ser la incorporación de sistemas expertos que ayuden en el establecimiento de prototipos y en la construcción del sistema.

Una tercera línea de investigación será la integración de los aspectos de producción de los sistemas con los aspectos de administración (planificación, control de versiones, control de calidad...).

Otro campo de investigación será el de obtención de herramientas formales de prueba y validación automática que cubran todo el ciclo de vida del sistema.

Asimismo puede llegar a ser interesante para la producción de software la utilización de técnicas multimedia en el entorno de más difícil formalismo y mayor creatividad, como es el establecimiento de las especificaciones.

Por último, a título de sugerencia, y sin agotar las posibilidades de investigación en este campo, que son amplísimas, se podría llevar a cabo una investigación en el aspecto de factoría de software, orientado a empresas que producen software para terceros, correspondientes a

distintos entornos (el presentado corresponde a un entorno de gestión). En este caso sería necesario fijar una serie de entornos de producción de software y asignar métodos, técnicas, formalismos y procedimientos que a la manera de mecano se emplearían para cada proyecto concreto.

BIBLIOGRAFIA

[ADL88] M. Adler. "*An Algebra for Data Flow Diagram Process Decomposition*". IEEE Transactions on Software Engineering, Vol. 14, no. 2, pp. 169-183, Feb.,1988.

[AEC86] AECC. "*Glosario de Términos de Calidad e Ingeniería del Software*". Asociación Española para el Control de Calidad,1988.

[ART87] Arthur Young Information Technology Group. "*The Arthur Young Practical Guide To Information Engineering*". John Wiley & Sons,1987.

[AVI88] D.E.Avision and G.Fitzgerald. "*Information Systems Development: Methodologies, Techniques and Tools*". Blackwell Scientific Publications,1988.

[BEL77] Bell, T., et al. "*An Extendable Approach to Computer-Aided Requeriments Engineering*". Trans. Software Eng., vol. SE-3, no.1, January,1977.

[BOE76] B.W. Boehm. "*Software Engineering*". IEEE Transactions on Computers, Vol. C-25, no. 12, pp. 1226-1241, December,1976.

[BOE81] B.W. Boehm. "*Software Engineering Economics*". Englewood Cliffs, NJ: Prentice-Hall,1981.

[BOE85] B.W. Boehm. "*The Basic Principles of Software Engineering: Part I,II*". Auerbach Publishers Inc., Systems Development Management. 33-11-10, 33-11-20, April/May,1985.

[BOH66] C. Bohm and G. Jacopini. "*Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules*". Communications of the ACM, Vol. 9, no. 5, pp. 366-371,1966.

[BOO86] G. Booch. "*Object-Oriented Development*". IEEE Transactions on Software Engineering, Vol. SE-12, no.2, pp. 211-221, Feb.,1986.

[BSP81] BSP-IBM. "*Business Systems Planning: Information Systems Planning Guide*". Ref. Manual: GE20-0527-2,1981.

[CHE76] P.P. Chen. "*The Entity-Relationship Model-Toward a Unified View of Data*". ACM Transactions on Database Systems, Vol. 1, no.1,

pp.9-36. March,1976.

[CHE79] P.P. Chen. "*E-R Approach to Systems Analysis & Design*". New York: North Holland,1979.

[CLU89] McClure,C. "*CASE is Software Automation*". Prentice-Hall,Englewood Cliffs,1989.

[COD72] E.F. Codd. "*Futher Normalization of the Data Base Relational Mode*". Data Base Systems. Randall Rustin ed, 1972.

[COL87] A. Collongues, J. Hugues, B. Laroche. "*Merise: Méthode de Conception*". Dunod Informatique, 1987.

[COL89] A. Collongues, J. Hugues, B. Laroche. "*Merise: Etudes et exercises*". Dunod Informatique, 1989.

[CUE87] G. Cuevas. "*Programación Estructurada: Diseño, Validación y Transformación de programas*". SEPA, Serie Informática,1987.

[CUE88] G. Cuevas. "*Una Herramienta para la Generación de Programas*". Dpto. Lenguajes, Sistemas Informáticos e Ingeniería de Software. Fac. de Informática de la U.P.M.,1988.

[CUE89] G. Cuevas Agustín. "*Ingeniería de Software: Diseño de Software*". Publicaciones de la Facultad de Informática de Madrid,1989.

[CUT87] G. Cutts. "*Structured Systems Analysis & Design Methodology*". Paradigm Publishing Ltd,1987.

[DAV88] A.M. Davis. "*A comparison of Techniques for the Specification of External System Behavior*". Communications of the ACM, Vol. 31, no. 9, pp. 1098-1115.

[DEM79] Tom DeMarco. "*Structured Analysis and Systems Specification*". Englewood Cliffs,N.J: Prentice-Hall,1979.

[DIC80] Brian Dickinson. "*Developing Structured Systems*". New York: Yourdon Press,1980.

[DIJ68] E.W. Dijkstra. "*Cooperating Sequential Processes*". Programming Languages. New York: Academic Press,1968.

[DIJ78] E.W. Dijkstra. "*Método de Programación: Objetivos y Naturaleza*". 1978.

[DOD87] DOD-STD-2167A. "*Draft Defense System Software Development*". Department Of Defense. USA, April,1987.

[DOW88] Downs,E.,Clare,P. and Coe,I. "*Structured Systems Analysis and Design Method: Aplication and Context*". Prentice-Hall, Hemel Hempstead,Herts,1988.

[EVA89] R. Rock Evans and B. Engelien. "*Analysis Techniques for CASE: a Detailed Evaluation*". Ed OVUM,1989.

[FAI87] Richard E. Fairlay. "*Software Engineering Concepts*". McGraw-Hill,1987.

[FLA81] M. Flavin. "*Fundamental Concepts of Information Modeling*". New York: Yourdon Press,1981.

[GAB89] J. Gabay. "*Apprendre et pratiquer Merise*". Masson, 1989.

[GAN79] Chris Gane and Trish Sarson. "*Structured Systems Analysis: Tools and Techniques*". Englewood Clifss,N.J.: Prentice-Hall,1979.

[GEH86] N. Gehani, A.D. Mc Gettrick. "*Software Specification Techniques*". Addison-Wesley,1986.

[GOL86] R. Goldberg. "*Software Engineering: An emerging discipline*". IBM Systems Journal, Vol. 25, nos. 3/4, pp.334-353, 1986.

[GUI86] GUIDE Publication GPP-147. "*Joint Application Design*". Chicago: GUIDE International,1986.

[HAN83] K. Hansen. "*Data Structured Program Design*". Ken Orr and Associates, Inc.,EEUU,1983.

[HOF85] G.F. Hoffnagle and W.E. Beregi. "*Automating the software development process*". IBM Systems Journal 24, no. 2, pp 102-120,1985.

[HUM85] W.S. Humphrey. "*The IBM large-systems Software Development Process: Objectives and direction*". IBM Systems Journal, Vol. 24, no. 2, 1985.

[HUN81] H. Hunk. "*Software Engineering Enviroments*". North-Holland,1981.

[IEE83] IEEE. "*Standard Glossary of Software Engineering Terminology*". IEEE Standard 729-1983.

[IEE84] IEEE. "*Guide to Software Requirements Specifications*". IEEE Standard 830-1984.

[IEE87] IEEE. "*Draft Standard for Software Rewiews and Audits*". IEEE Standard P1028,1987.

[JAC75] Michael Jackson. "*Principles of Program Design*". New York: Academic Press,1975.

[JAC83] Jackson,M.A.. "*Systems Development*". Prentice-Hall, New Jersey,1983.

[KIN83] David King. "*Current Practices in Software Engineering*". New York: Yourdon Press,1983.

[KRA88] J. Kramer and Keng Ng. "*Animation of Requirements Specifications*". Software-Practice and Experience, Vol.18, no. 8, pp 749-774, August,1988.

[KUN89] Kung. "*Conceptual modeling in the context of Software Development*". IEEE Transactions on Software Engineering,vol.15,no.10,pp1176-1187,Oct.1989.

[LAW88] D. Law. "*Methods for Comparing Methods*". NCC Publications, 1988.

[MAA88] S. Mass, M.B. Rosson and W.A. Kellogg. "*The Designer as User: Building Requirements for Design Tools from Design Practice*". Communications of the ACM, Vol. 31, no. 11, November,1988.

[MAR81] Martin,J. and Finkelstein,C. "*Information Engineering*". Volumens 1-2. Prentice-Hall, Englewood Cliffs,1981.

[MAR85] J. Martin and C.Mc. Clure. "*Diagramming Techniques for Analyst and Programmers*". Prentice-Hall,1985.

[MAR87] J.T. Martínez Armero. "*Ingeniería de Requerimientos en el Ciclo de Vida de Software:*

una Metodología Formal de Análisis y derivación de Requerimientos". Tesis Doctoral presentada en la Facultad de Informática de Madrid de la U.P.M., Enero,1987.

[MAT82] L. Maté Hernández. "*El Ciclo de Vida de un Sistema Informático*". Facultad de Informática de Madrid. Departamento de Publicaciones, 1982.

[MAT86] L. Maté Hernández. "*Planificación y Gestión de desarrollo de sistemas informáticos*". Facultad de Informática de Madrid. Departamento de Publicaciones, 1986.

[MAT87] K. Matsumara, H. Mizutani, M. Arai. "*An application of Structural Modeling to Software Requirements Analysis and Design*". IEEE Transactions on Software Engineering, Vol. 13, no. 4, pp. 461-471, April,1987.

[MAY85] R.G.Mays, L.S.Orzech, W.A.Ciarfella, R.W.Philips. "*PDM: A requirements methodology for software system enhancements*". IBM Systems Journal, Vol.24, no.2,pp 134-149,1985.

[MCU89] D.J. McCubbey, P.O. Flaatten, P.D. O'Riordan, K. Burgess. "*Foundations of Bussiness Systems*". The Dryden Press,1989.

[MEN79] K. Mensah. "*Criteria for Decomposing an Information System into its Subsystems for Business Systems Planning*". IBM, Los Angeles Scientific Center. Tec. Rep. G320-2700, March,1979.

[MEN80] K. Mensa. "*Computer-Aided Modeling and Analysis Techniques for Determining Management Information Systems Requirements*". IBM, Los Angeles Scientific Center. Tec. Rep. G320-2703, March,1980.

[MEN82] Stephen McMenamin & John Palmer. "*The Transition Between Analysis and Design*". GUIDE 54 International Corporation,1982.

[MEN84] Stephen McMenamin & John Palmer. "*Essential Systems Analysis*". New York: Yourdon Press,1984.

[MET83] P.W. Metzger. "*Managing a Programming Project 2nd ed.*". Englewood Cliffs NJ: Prentice-

Hall,1983.

[NCC86a] NCC. "*SSADM. Volume 1: Tasks and Terms*". NCC, 1986.

[NCC86b] NCC. "*SSADM. Volume 2: Techniques and Documentation*". NCC, 1986.

[MIL87] H.D. Mills, R.C. Linger, A.R. Herner. "*Box Structured Information Systems*". IBM Systems Journal, Vol. 26, no. 4, pp. 395-413, 1987.

[ORR77] Ken Orr. "*Structured Systems Development*". New York: Yourdon Press,1977.

[ORZ79] L.S. Orzech. "*PSL/PSA: A computer-aided tool and technique for specification and analysis of high-level designs*". IBM/FSD Software Engineering Exchange 2, no.1, pp 2-9, October,1979.

[PAG88] M. Page-Jones. "*The Practical Guide to Structured Systems Design*". 2nd ed. Englewood Cliffs,N.J.: Prentice-Hall,1988.

[PET88] Lawrence Peters. "*Advanced Structured Analysis and Design*". Prentice-Hall,1988.

[PHA89] Pham Thu Quang. "*Merise Appliquée*". Eyrolles, 1989.

[PRE88] R. S. Pressman. "*Software Engineering: A Practitioner's Approach 2nd ed.*". Mc.Graw-Hill,1988.

[RAM84] Ramamoorthy. "*Software Engineering problems and perspectives*". Computer,vol.17,no.10,pp191-209,Oct.1984.

[ROS75] D.A. Ross. "*Software Engineering: Process, Principles, and Goals*". Computer, May,1975.

[ROS77] D. Ross. "*Structured Analysis (SA): A Language for Communicating Ideas*". Trans. Software Eng., vol. SE-3, no.1, January,1977.

[ROS79] D. Ross and K. Schoman. "*Structured Analysis for Requirements Definition*". IEEE Transactions on Software Engineering, Volume SE-3, Number 1,pp.6-15,January 1977.

[ROS88] M.B. Rosson, S. Maass, W.A. Kellogg. "*The Designer as user: Building Requirements for Design Tools from Design Practice*". Communications of the ACM, Vol. 31, no. 11, pp. 1288-1297, Nov.,1988.

[SHE81] S.E. Shepard and B. Curtis. "*The Effects ofSpatial Arrangement on the Comprehension of Software Specifications*". International Conference on Software Engineering, IEEE, March, 1981, pp. 207-214.

[SHO83] Shooman. "*Software Engineering: Design, Reliability and Management*". Mc.Graw-Hill,1983.

[SHO87] P. Shoral, M. Even-Chaime. "*Data Base Schema Design: An Experimental Comparison Between Normalization and Information Analysis*". Data Base, Vol. 18, no. 3, pp. 30-39, Spring,1987.

[STA87] STARS. "*The STARTS Guide Second Edition*". NCC Publications, Vol. 1-2, 1987.

[STA88] STARS. "*The STARTS Purchasers' Handbook*". NCC Publications, 1988.

[STE74] W. Stevens, G. Myers, and L. Constantine. "*Structured Design*". IBM Systems Journal, Vol. 13, no. 2, May,1974.

[STE82] W.P. Stevens. "*How data flow can improve application development productivity*". IBM Systems Journal, Vol. 21, no. 2, 1982.

[TAR89a] H. Tardieu, A. Rochfeld, R. Colletti. "*La Méthode Merise: Principles et Outils*". Les Éditions D'Organisation, 1989.

[TAR89b] H. Tardieu, A. Rochfeld, R. Colletti. "*La Méthode Merise: Démarche et pratiques*". Les Éditions D'Organisation, 1989.

[TEI77] D. Teichrow and E. Hershey. "*PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems*". Trans. Software Eng., vol. SE-3, no.1, January,1977.

[TSE77] IEEE. "*Special Issue on Requirements Analysis*". IEEE Trans. Software Eng., vol. SE-3, no.1, January,1977.

[VEG85] C. Vega, G. Cuevas, I. Vega. "*Sobre algunos aspectos de Automatización de Programación Estructurada*". Facultad de Informática. U.P.M.,1985.

[WAR80] J.D. Warnier. "*Les Procedures de traitement et leurs données*". Les Editions D'Organization. Paris,1980.

[WAR81] J.D. Warnier. "*Logical Construction of Systems: Warnier-Orr Diagrams*". Van Nostrance Reinhold Company, pp. 11-38, New York,1981.

[WAR84] Paul Ward. "*Systems Development Without Pain*". New York: Yourdon Press,1984.

[WAR85] J.D. Warnier. "*Entrainement a la programmation LCP*". Les Editions D'Organization. Paris,1985.

[WAR85] Paul Ward and Stephen Mellor. "*Structured Development for Real-Time Systems*". Volumens 1-3. New York: Yourdon Press,1985.

[WAS81] A.I. Wasserman. "*The Ecology of Software Development Enviroments*". Software Development Enviroments, pp. 47-52,1981.

[WEI84] G.M. Weinberg and D.E. Freedman. "*Rewiews, Walkthroughs and Inspections*". IEEE Transactions on Software Engineering, no. 1,January,1984.

[YAD88] S.B. Yadar, R.R. Bravoco, A.t. Chatfiel, T.M. Rajkumar. "*Comparison of Analysis Techniques for Information Requirement Determination*". Communications of the ACM, Vol. 31, no. 9, pp 1090-1097, Sep,1988.

[YOU79] Edward Yourdon and Larry Constantine. "*Structured Design*". Englewood Cliffs,N.J.: Prentice-Hall,1979.

[YOU88] Edward Yourdon. "*Managing the System Life Cicle*". 2nd ed. New York: Yourdon Press,1988.

[YOU89] Edward Yourdon. "*Structured Walkthroughs 4ª Edition*". Yourdon Press,1989.