# CAPITULO I

# MARCO TEORICO Y CONCEPTUAL

## 1.1 Antecedentes

El Departamento de Recursos Humanos del Hospital Nacional de Niños Benjamín Bloom realiza diversas actividades que se relacionan entre si, desde la publicación de plazas de trabajo y los procesos de contratación hasta las gestiones administrativas de renuncias y despidos. Durante el tiempo que el empleado labora dentro de la Institución, se realizan muchas gestiones administrativas que generan una infinidad de flujos, procesos y almacenamientos de datos que hasta la fecha se han realizado manualmente en un ochenta por ciento, un diez por ciento en software diseñado para administración de hojas electrónicas y otro diez por ciento en aplicaciones aisladas incompletas y poco confiables.

Debido a que todas estas actividades se pueden sistematizar, siempre se han hecho esfuerzos por organizarlas de tal manera que llevarlas a cabo no sea objetable sino mas bien eficaz y confiable. Lastimosamente los esfuerzos que se han hecho no han tenido mayor impacto en el funcionamiento del departamento, esto se debe a la falta de un software multiusuario que constituya una herramienta fundamental para estructurar en forma lógica y adecuada el procesamiento de los datos y así lograr los objetivos planteados.

El personal que labora en la gestión administrativa del recurso humano está conciente de esta realidad y comprenden que una Institución de servicio tan importante como es el Hospital Bloom, debe buscar la modernización y la funcionabilidad de los sistemas. Con referencia al equipo existente para el proyecto, podemos hacer uso de siete computadoras ya existentes que pueden constituirse en una red de área local, también se cuenta con impresores adecuados como dispositivos de salida.

Es evidente la necesidad de desarrollar un sistema de procesamiento electrónico de datos con técnicas modernas orientadas a objetos, y luego de la implementación, que éste pueda ser utilizado por el personal del departamento de recursos humanos para satisfacer todos los requerimientos de información operativa, para mandos medios, gerencia y entidades externas.

Las áreas que con urgencia deben tener procesos mecanizados son reclutamiento, contratación y seguimiento de empleados. Cabe mencionar que en el proceso de selección serán considerados solo algunos aspectos relevantes sin incluir el procesamiento de evaluaciones y selección automatizada. Esto se debe a que en la selección predominan muchos aspectos de criterio personal por parte de las Jefaturas. Sin embargo, nuestro sistema proporcionará elementos valiosos para la toma de decisión respecto a la selección de personal a contratar.

## 1.2 Teorías Fundamentales

A continuación se presentan los fundamentos teóricos a partir de los cuales se analizarán las hipótesis o los planteamientos del proyecto.

### 1.2.1 Análisis, Diseño y Construcción de Sistemas de Información

Un sistema es un conjunto de componentes que interaccionan entre si para lograr un objetivo común.

Dentro de las organizaciones el desarrollo de sistemas ha permitido mejorar los métodos y procedimientos.

El desarrollo de sistemas de información esta compuesto de tres componentes fundamentales: El análisis del sistema, el diseño del sistema y la construcción del software o aplicación.

Los sistemas emplean un modelo de control básico que consiste en:

- Un *estándar* para lograr un desempeño aceptable.
- Un método para *medir* el desempeño actual.
- Un método para *comparar* el desempeño actual contra el estándar.
- Un método para la *retroalimentación*

El diseño de sistemas es el proceso de planificar reemplazar o complementar un sistema organizacional existente. Pero antes de llevar a cabo esta planeación, es

necesario comprender en su totalidad el viejo sistema y determinar la mejor forma en que se pueden, si es posible, utilizar las computadoras para hacer la operación más eficiente. El análisis de sistemas, por consiguiente, es el proceso de clasificación e interpretación de los hechos, diagnóstico de problemas y empleo de la información para recomendar mejoras al sistema. Luego, la programación o la creación del software permitirá la mecanización de los procesos cuya factibilidad sea posible.

Los sistemas de información se clasifican en las siguientes categorías: [1]

*Sistema para el Procesamiento de Transacciones:* Sustituye los procedimientos manuales por otros basados en computadora. Trata con procesos de rutina bien estructurados. Incluye aplicaciones para el mantenimiento de registros.

*Sistema de Información Administrativa:* Proporciona la información que será empleada en los procesos de decisión administrativos. Trata con el soporte de situaciones de decisión bien estructuradas. Es posible anticipar los requerimientos de información más comunes.

*Sistema para el Soporte de Decisiones:* Proporciona información a los directivos que deben tomar decisiones sobre situaciones particulares. Apoyan la toma de decisiones en circunstancias que no están bien estructuradas.

[1] *Análisis y Diseño de Sistemas de Información. James A. Senn. Pag.32*

Existen cuatro enfoques al desarrollo de Sistemas de Información basados en computadora:

- Método del Ciclo de Vida para el Desarrollo de Sistemas.
- Método del Desarrollo del Análisis Estructurado.
- Método del Prototipo de Sistemas.
- Modelo Multiparte Orientado a Objetos.

*Método del Ciclo de Vida para el Desarrollo de Sistemas*

Este método esta compuesto del conjunto de actividades que los analistas, diseñadores y usuarios realizan para desarrollar e implementar un sistema de información. Este método consta de las siguientes actividades:[2]

- ✓ Investigación preliminar.
- ✓ Determinación de los requerimientos.
- ✓ Diseño del sistema.
- ✓ Desarrollo de software.
- ✓ Prueba de los sistemas.
- ✓ Implantación y Evaluación.

*Método del Desarrollo del Análisis Estructurado*

El análisis estructurado se concentra en especificar lo que se requiere que haga el sistema o la aplicación. No se establece cómo se cumplirán los requerimientos o la

[2] *Análisis y Diseño de Sistemas de Información. James A. Senn. Pag.33*

forma en que se implantará la aplicación. Mas bien permite que las personas observen los elementos lógicos (lo que hará el sistema) separados de los componentes físicos (computadoras, terminales, sistemas de almacenamiento, etc.).

Después de esto se puede desarrollar un diseño físico eficiente para la situación dada y el lugar donde será utilizado. Los elementos para el diseño estructurado son:[3]

- ✓ Descripción Gráfica.
- ✓ Diagramas de Flujo de Datos.
- ✓ Diccionario de Datos.

*Herramientas Asistidas por Computadora para la Ingeniería de Sistemas (CASE)*

Las siglas CASE (Computer Aided Software Engineering) se emplean con bastante frecuencia para denotar la ingeniería de sistemas asistida por computadora o para ser más exacto, Ingeniería del Software Asistida por Computadora.

Las herramientas de tipo CASE incluyen los siguientes cinco componentes:

- ✓ *Herramientas para la Diagramación*
- ✓ *Depósito Centralizado de Información*
- ✓ *Generador de Interfaces*
- ✓ *Generadores de Códigos*
- ✓ *Herramientas de Administración*

[3] *Análisis y Diseño de Sistemas de Información. James A. Senn. Pag.40*

*Método del Prototipo de Sistemas*

Este método hace que el usuario participe de manera más directa en la experiencia de análisis y diseño que cualquiera de los ya presentados. La construcción de prototipos es muy eficaz bajo las circunstancias correctas. Sin embargo, al igual que los otros métodos, el método es útil solo si se emplea en el momento adecuado y en la forma apropiada.

El prototipo es un sistema que funciona desarrollado con la finalidad de probar ideas y suposiciones relacionadas con el nuevo sistema. Al igual que cualquier sistema basado en computadora, está constituido por software que acepta entradas, realiza cálculos, produce información ya sea impresa o presentada en una pantalla, o que lleva a cabo otras actividades significativas. Es la primera versión, o iteración, de un sistema de información; es el modelo original.

*Modelo Multiparte Orientado a Objetos*

Todas las cosas de la naturaleza que nos rodean son objetos y por eso no debe sorprendernos que se ponga una visión orientada a objetos para la creación de software de computadora, una abstracción que modela el mundo de forma tal que nos ayuda a entenderlo y gobernarlo mejor.

A medida que pase el tiempo las tecnologías de objetos sustituirán a los enfoques clásicos de desarrollo de software, esto se debe a que las tecnologías de objeto llevan a reutilizar y reutilización (de componentes de software), lleva a un desarrollo de

software más rápido y  a programas de mejor calidad. El software orientado a objetos es mas fácil de mantener y escalar  debido a que su estructura es inherentemente descompuesta.

*Selección del Software para la codificación de la Aplicación*

Una de las tareas más difíciles en el desarrollo de sistemas es la selección del software para construir la aplicación, una vez que se conocen los requerimientos del sistema, se debe determinar si un cierto paquete de software o lenguaje de programación cumple con los requerimientos.

*Cronología de los Lenguajes de Programación*

Al principio, IBM en sus primeros años, creó el primer Sistema de Programación Automática, que llamó Ensamblador. Entonces los programadores ya podían hablar de instrucciones de máquina:  MOV ax, bp ;  en lugar de secuencias binarias de programación:

1000100111100011001001101100011011001110000 1.

Los computadores al principio eran tan caros que sólo los militares los podían pagar, por lo que las primeras aplicaciones computacionales fueron el cálculo numérico de complejas ecuaciones.

Cuando surgieron los computadores que permitían la programación interactiva nació BASIC, y con él los números de línea.

La década de los sesenta, produjo lenguajes como Pascal, Algol y PL/I, que difieren de BASIC en que permiten el uso de argumentos, variables locales y programación estructurada. Pascal llena muy bien el paradigma de la programación estructurada, por lo que se ha usado mucho, principalmente en Universidades.

Conforme aumentó la capacidad de cómputo de los computadores, y se redujo su costo, surgieron nuevos lenguajes que permitían aplicar la computación a nuevos campos.

Así nacen una gran cantidad de nuevos desafíos tecnológicos, muchos de los cuales todavía no han sido adecuadamente resueltos. Las últimas tres modas en la tecnología de programación, Programación Lógica, Lenguajes de Cuarta Generación (4GLs) y Programación Orientada a Objetos, son respuesta a estas nuevas aplicaciones.

Necesariamente las nuevas tecnologías son producto de las viejas. Las ideas que dieron vida a Pascal no pudieron haber existido sin COBOL; para entender una nueva herramienta es importante entender cómo fue creada. Además, el uso de nuevas herramientas rápidamente lleva a detectar sus defectos. Por eso en el nuevo lenguaje se trata de arreglar los defectos de sus antecesores.

*Lenguajes de Cuarta Generación (4GLs)*

El lenguaje COBOL representa un avance tecnológico muy importante. Por mucho tiempo la mayor área de aplicación de las computadoras ha estado en los ambientes de negocios, en el proceso de datos relevantes a actividades administrativas. Por eso nació

COBOL y tuvo una gran aceptación en las máquinas grandes ("mainframes"). Sin embargo, en COBOL hay que escribir mucho código para programar: en COBOL uno no dice leche, dice "líquido perlático de la consorte del toro, pero de la que estuvo recientemente preñada".

Este defecto del COBOL por ser anacrónico, indujo a muchas casas productoras de programas a crear herramientas que permitieran programar con más comodidad.

Con el desarrollo de la tecnología de bases de datos fue posible identificar los siguientes componentes de los programas en Sistema de Información:

- ✓ Diálogos por pantalla y controles de acceso.
- ✓ Manipulación de archivos y bases de datos.
- ✓ Reportes.

Los 4GLs vienen a solventar el problema de expresividad de COBOL, y luego se convierten en herramientas para programar cada uno de los componentes arriba mencionados. Por eso los primeros 4GLs eran generadores de código COBOL, y los modernos son ambientes interactivos de desarrollo de sistemas, cargados de editores muy especializados. Lo común a todos los 4GLs es que minimizan la cantidad de código que hay que escribir para obtener un Sistema de Información.

Aunque al principio los 4GLs tenían algunas restricciones, los actuales son una respuesta satisfactoria para programar Sistemas de Información. Permiten ensamblar

aplicaciones involucrando al futuro usuario del sistema, y consumen relativamente pocos recursos.

Los 4GL han permitido también crear sistemas usando prototipos. Un prototipo es un esqueleto, que luego un programador hábil toma y completa con código. De esta manera el analista, en lugar de hablar o de escribir especificaciones, se sienta con su usuario ante una pantalla, y comienza a construir, junto a él, el Sistema de Información.

La gran ventaja del uso de prototipos es que el usuario puede participar (y quejarse) en las primeras etapas del diseño del sistema, con lo que se abarata mucho el costo total del sistema.

*Programación Orientada a Objetos*

La programación orientada a objetos será eficiente si y solo si se hizo un buen análisis y diseño, de hecho el producto final dependerá de estas dos fases del desarrollo de sistemas. Para construir el programa se han creado diferentes lenguajes de programación orientados a objetos que pueden ser utilizados para este fin, entre ellos podemos mencionar los siguientes:

- ✓ *Lenguaje de programación Visual Basic*
- ✓ *Lenguaje de Programación Visual Fox*
- ✓ *Lenguaje C Orientado a Objetos*
- ✓ *Lenguaje Java*
- ✓ *Power Builder*

*Gestor de Base de Datos*

Una base de datos suele definirse como un conjunto de información organizada en tablas y relacionada con un asunto o con una finalidad, tal como el seguimiento de los pedidos de clientes o una colección de música.

Una tabla es un conjunto de registros sobre un tema específico, como productos o proveedores.

En la terminología propia de las bases de datos hay tres conceptos claves dentro de las tablas: Dato, Campo y Registro (como se muestra en la Figura 1).

Dato: Es la unidad mínima de información que se puede tener sobre una persona o cosa específica. Cualquier información particular como por ejemplo fecha de nacimiento de un empleado se considera un dato.

Campo: Es un nombre genérico que se asigna para indicar o representar un dato o un conjunto de datos en una tabla. Cada campo contiene un fragmento de información que describe una parte única de un registro.

Registro: Es un conjunto de campos relacionado con un mismo ente de Información, así por ejemplo, los datos personales del empleado forman un registro.

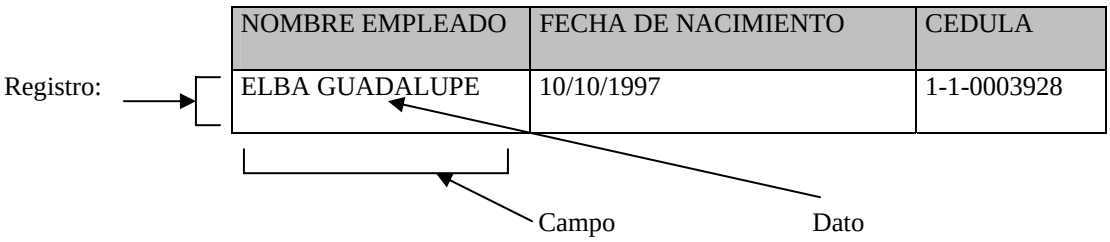



Figura 1

Entre los gestores de base de datos podemos mencionar los siguientes:

- ✓ *Gestor de Base de Datos* Microsoft Access
- ✓ *Gestor de Base de Datos ORACLE*
- ✓ *Gestor de Base de Datos SYBASE*
- ✓ *Gestor de Base de Datos SQL (Structured Query Language)*

## 1.3 Adopción de la Teoría

### 1.3.1 Modelado del Proceso Orientado a Objetos

Durante muchos años el término orientado a objetos (OO) se usó para significar un enfoque de desarrollo de software que usaba uno de los lenguajes orientados a objetos. Hoy en día el paradigma OO encierra una completa visión de la ingeniería del software.

Los beneficios de la tecnología orientada a objetos se fortalecen si se usa antes y durante el proceso de ingeniería del software.

Esta tecnología orientada a objetos considerada debe hacer sentir su impacto en todo el proceso de ingeniería del software. Un simple uso de programación orientada a objetos(POO) no brindará los mejores resultados. Los Ingenieros del software y sus directores deben considerar estos elementos como Análisis de requisitos Orientado a Objetos (AOO), Diseño Orientado a Objetos(DOO), Análisis del Dominio Orientado a Objetos (ADOO), Sistemas de Gestión de Base de Datos Orientadas a Objeto (SGBDOO), Programación Orientada a Objetos, Pruebas Orientadas a Objetos y de Ingeniería del Software Orientada a Objetos Asistida por Computadora (ISOOAC).[4] Como se muestra (Figura 2), el modelo de proceso de ensamblaje de componentes ha sido empotrado por la ingeniería del software orientado a objetos.

MODELADO PARA DESARROLLO ORIENTADO A OBJETOS

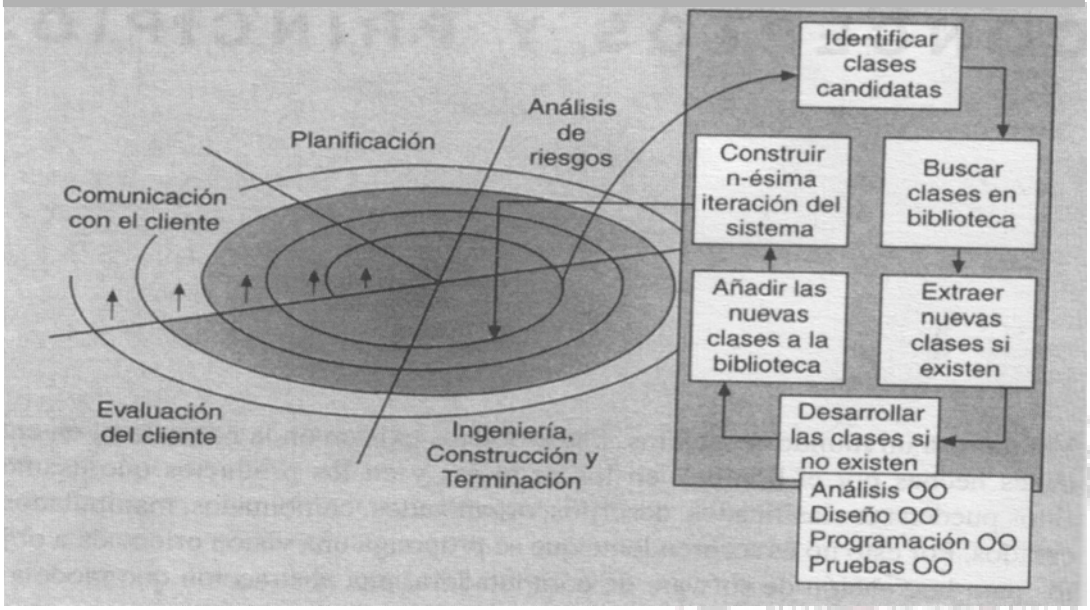



Figura 2

[4] *Ingeniería del Software. Roger S. Pressman. Pag.638*

En el modelado del proceso OO encontramos los siguientes elementos:

*Clases: E*ncapsulan las abstracciones de datos y procedimientos que se requieren para describir el contenido y comportamiento de alguna entidad del mundo real. Puede decirse que una clase es una descripción generalizada (por ejemplo, una plantilla, un patrón o una copia) que describe una colección de objetos similares. Por definición, todos los objetos que existen dentro de una clase heredan sus atributos y las operaciones disponibles para la manipulación de los atributos. Una superclase es una colección de clases y una subclase es una instancia de una clase.

*Objeto:* Es una entidad física (casa, auto, etc.), pero también puede ser un concepto (ecuación matemática). En modelado OO es una unidad que integra estado y comportamiento.

*Atributos:* Los atributos están asociados a las clases y objetos, estos describen la clase o el objeto de alguna manera. Un atributo puede tomar un valor definido de un dominio, o sea un rango de valores probables a tomar. Podemos decir que un atributo es un campo que guardará un valor específico.

*Dominio:* Son todos los valores que puede contener un atributo, por ejemplo un dominio para profesión puede ser: Ingeniero, Médico, Enfermera, etc. En situaciones mas complejas un dominio puede ser un conjunto de clases.

*Operaciones, Métodos y Servicios:* Las operaciones son encapsuladas junto con los datos en un objeto. Los algoritmos que procesan estos datos se llaman: Operaciones,

Métodos o Servicios (módulos). Cada operación representa un comportamiento (contratar, pagar, etc.) del objeto y puede proporcionar un estimulo (mensajes) que reaccionan en cadena con otros objetos para satisfacer el comportamiento demandado.

*Mensajes:* Los mensajes son el medio a través del cual los objetos interactúan. Un mensaje estimula la ocurrencia de cierto comportamiento en el objeto receptor. El comportamiento se realiza cuando se ejecuta una operación.

*Encapsulamiento, Herencia y Polimorfismo:* El encapsulamiento de datos y de operaciones se da dentro de los objetos y luego en las clases, éstas a su vez dentro de un mismo paquete, esto permite importantes beneficios como son:

- Los detalles de implementación interna de datos y procedimientos están ocultos al mundo exterior. Esto reduce la propagación de efectos colaterales cuando ocurren cambios.

- Las estructuras de datos y las operaciones que las manipulan están mezcladas en una entidad sencilla: la clase. Esto facilita la reutilización de componentes (de software).

- Las interfaces entre objetos encapsulados están simplificadas. Un objeto que envía un mensaje no se tiene que preocupar de los detalles de estructuras de datos internas en el objeto receptor.

La herencia es una de las diferencias clave entre sistemas convencionales y sistemas orientados a objeto (Figura 3). Una herencia Y hereda todos los atributos y operaciones asociadas con su superclase X. Esto significa que todas las estructuras de datos y algoritmos originalmente diseñados e implementados para X están inmediatamente disponibles para Y. La reutilización se realiza directamente.

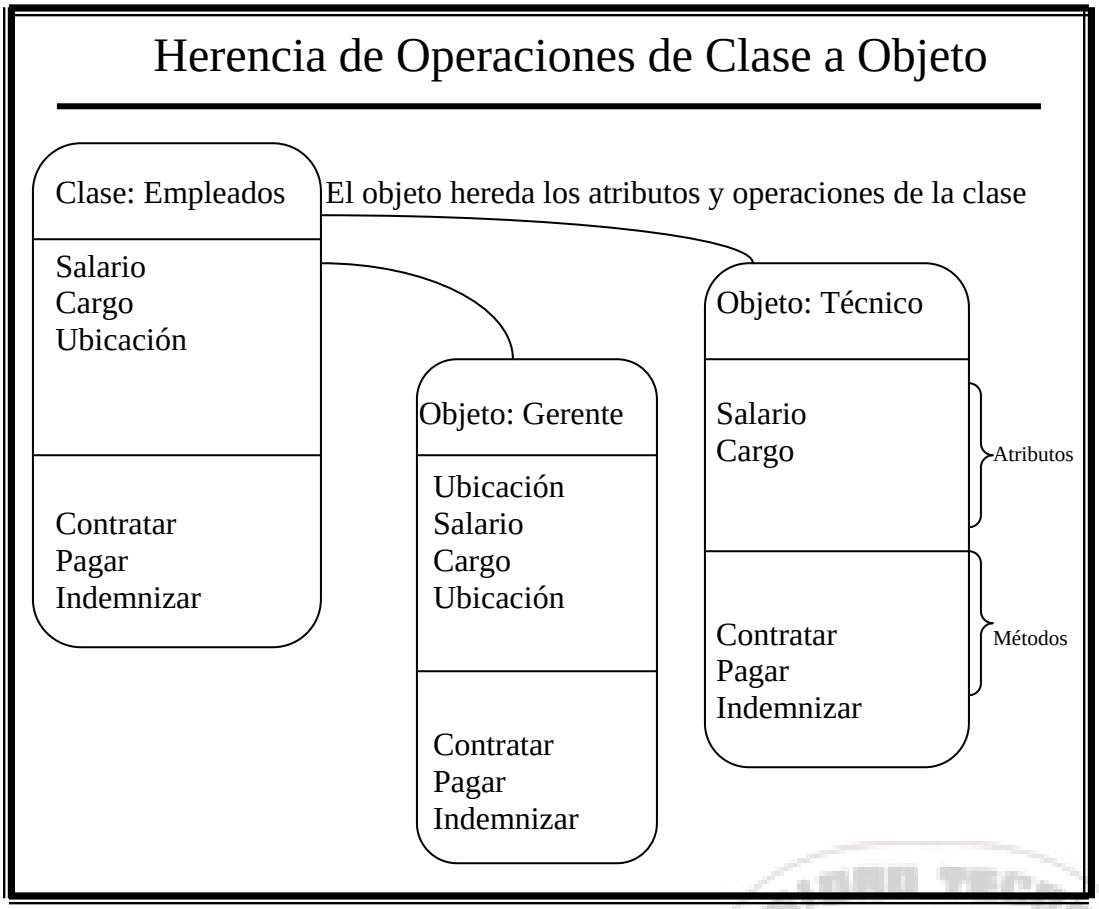



Figura 3

El polimorfismo es una característica que reduce en gran medida el esfuerzo necesario para extender un sistema orientado a objetos. Este permite que un número de operaciones diferentes tengan el mismo nombre. Esto reduce el acoplamiento entre objetos, haciendo a cada uno mas independiente.

*Identificación de las clases y objetos como elementos de un modelo de Objetos*

En una habitación existen un conjunto de objetos físicos que pueden ser fácilmente identificables, clasificados y definidos. Pero cuando se observa el espacio de un problema en una aplicación de software, los objetos pueden ser mas difíciles de identificar.

Podemos identificar objetos examinando el planteamiento del problema (descripción o narrativa completa en forma textual, de todo el sistema que se está analizando). Se determinan subrayando cada nombre o cláusula nominal e introduciéndola en una tabla simple (Objeto/Clase potencial – Tipo de Objeto). Una vez aislados los nombres, buscamos los tipos de objetos. Estos pueden ser:[5]

- Entidades externas: Son otros sistemas, dispositivos o personas que producen o consumen información por un sistema computacional. Ej. Sistema Contable, Sistema de Pagos, Dirección, Gerencia, etc.

- Cosas: Son informes, presentaciones, cartas y señales que son parte del dominio de información del problema. Ej. Nomina de empleados, Constancias de tiempo de trabajo, contratos, etc.

- Ocurrencias o Eventos: Pueden ser transferencias de propiedad o la terminación de una serie de movimientos de un robot que ocurren dentro del contexto de operación del sistema. Ej. Renunciar, Contratar, Jubilarse, etc.

[5] *Ingeniería del Software. Roger S. Pressman. Pag. 375*

- Papeles o Roles: Puede ser un director, ingeniero o un vendedor, o sea que es el desempeño que pueden tener las personas que interactúan con el sistema.

- Unidades Organizacionales: Son las divisiones, grupos o equipos que son relevantes en una aplicación. Ej. Area médica, Técnica, Administrativa, etc.

- Lugares: Este puede ser la planta de producción o muelle de carga que establece el contexto del problema y la función general del sistema. Ej. Departamento de Recursos Humanos, Departamento Financiero Contable, etc.

- Estructuras: Estos son sensores, vehículos de cuatro ruedas, computadoras, etc que definen una clase de objetos.

Después de clasificar los objetos como objeto potencial, debemos reclasificarlos de tal manera que puedan formar clases y ser parte del modelo de análisis. Para lograrlo estos deben poseer las siguientes características [6]:

1. Información retenida: La información acerca de él debe recordarse para que el sistema funcione.

2. Servicios necesarios: Debe poseer un conjunto de operaciones identificables que pueden cambiar de alguna manera el valor de sus atributos. Ej. Contratar, Pagar, Indemnizar, etc.

[6] Ingeniería del Software. Roger S. Pressman Pag. 378.

3. Atributos múltiples: Un objeto puede tener un solo atributo en la fase de análisis, pero en la parte del diseño, este atributo puede ser parte de otro objeto.

4. Atributos comunes: Los atributos que se definen para un objeto deben ser aplicables a todas las ocurrencias del objeto.

5. Operaciones comunes: Las operaciones que se definen para un objeto deben ser aplicables a todas la ocurrencias del objeto.

6. Requisitos esenciales: Serán definidas como objetos todas las entidades externas que producen o consumen información esencial para la producción de cualquier solución para el sistema.

Para ser considerado un objeto válido a incluir en el modelo de requisitos, un objeto potencial debe satisfacer todas o casi todas las características anteriores.

*Especificación de los atributos de un objeto*

Los atributos son los que definen al objeto, que clarifican lo que se representa con el objeto en el contexto del espacio del problema. Para cada objeto debe responderse la siguiente interrogante: ¿Qué elementos (compuestos y/o simples) definen completamente al objeto en el contexto del problema actual?

A partir de la narrativa o descripción del sistema nos respondemos esta pregunta y definimos los atributos que tiene cada objeto.

*Definición de Operaciones*

Una operación cambia valores de uno o mas atributos contenidos en el objeto. Estas operaciones pueden clasificarse en tres grandes categorías:

- ✓ Operaciones que manipulan los datos (añadir, eliminar seleccionar, etc)
- ✓ Operaciones que realizan cálculos.
- ✓ Operaciones que monitorizan un objeto frente a la ocurrencia de un suceso de control.

Nuevamente debemos revisar la descripción del sistema y en su forma gramatical debemos aislar los verbos que indican la conexión de objetos para definir las operaciones.

*Marco del Proceso común para OO (MPC)*

MPC Define un enfoque organizativo para el desarrollo y mantenimiento del software. Identifica el paradigma de ingeniería del software aplicado para construir y mantener el software, así como las tareas, hitos y entregas requeridos.

Por naturaleza, la ingeniería del software orientada a objetos debe contemplar un paradigma que contemple un desarrollo iterativo o sea que el software evoluciona a través de un número de ciclos. Cada ciclo esta compuesto de fases y cada fase tiene un

numero de iteraciones. Cada iteración del proceso recursivo/paralelo requiere ingeniería (análisis, diseño, extracción de clases, prototipo y pruebas). Durante las primeras etapas del proceso de ingeniería, el análisis y el diseño ocurren iterativamente. La intención es aislar todos los elementos importantes del análisis OO y de los modelos de diseño. Al continuar el trabajo de ingeniería, se producen versiones incrementales del software. Durante la evaluación se realizan para cada incremento revisiones, evaluaciones del cliente y pruebas, las cuales producen una retroalimentación que afecta a la siguiente actividad de planificación y el subsiguiente incremento.

Algoritmo de funcionamiento:

Inicia el Ciclo de vida

Fase 1: Estudio de oportunidad.

Fase 2: Elaboración.

Fase 3: Construcción.

(1-n)Iteraciones: Análisis, Diseño, Codificación, Pruebas e Integración

Fase 4: Transición.

(1-n)Iteraciones: Análisis, Diseño, Codificación, Pruebas e Integración

Mostrar nueva versión del producto

Si es aceptada: Ir a Fin de Ciclo

Volver al inicio de Ciclo

Fin de Ciclo (Aplicación Terminada)

La descripción de cada fase se presenta a continuación:

*Estudio de Oportunidad (Inception):* Define el ámbito y objetivos del proyecto; además se define la funcionabilidad y capacidades del producto.

*Elaboración:* Se estudian en profundidad la funcionalidad como el dominio del problema. Se define una arquitectura básica y se planifica el proyecto considerando recursos disponibles.

*Construcción:* En esta fase se desarrolla el producto a través de iteraciones donde cada iteración involucra tareas de análisis, diseño e implementación. Aquí se refina de manera incremental conforme se construye (Se permiten cambios en la estructura) y una buena parte del trabajo es de programación y pruebas; además se documenta el producto.

*Transición:* Se libera el producto y se entrega al usuario para su uso real, luego se incluyen tareas de empaquetado atractivo, instalación, configuración, entrenamiento, soporte, mantenimiento, etc. y se complementan los manuales de usuario.

Aunque el modelo de proceso recursivo/paralelo es el mejor marco de trabajo para un proyecto OO, el paralelismo dificulta el seguimiento del proyecto. Puede tenerse problemas para establecer los hitos significativos en un proyecto debido a que cierto numero de cosas están ocurriendo a la vez.

En general, los siguientes hitos pueden considerarse completados al cumplirse los criterios mostrados:[7]

Hito técnico: Análisis Orientado a Objetos terminado

- Todas las clases, la jerarquía de clases, están definidas y revisadas.
- Los atributos de clases y las operaciones asociadas a una clase se han definido y revisado.
- Las relaciones entre clases se han establecido y revisado.
- Se ha creado y revisado un modelo de comportamiento.
- Se han marcado clases reutilizables.

Hito técnico: Diseño Orientado a Objetos terminado

- El conjunto de subsistemas se ha definido y revisado.
- Las clases se han asignado a subsistemas y han sido revisadas.
- Se ha establecido y revisado la asignación de tareas.
- Se han identificado responsabilidades y colaboraciones.
- Se han diseñado y revisado los atributos y operaciones.
- El modelo de mensajería(paso de mensajes) se ha creado y revisado.

Hito técnico: Programación Orientada a Objetos terminada

- Cada nueva clase ha sido implementada en código a partir del modelo de diseño.
- Las clases extraídas se han integrado.

[7] *Ingeniería del Software. Roger S. Pressman. Pag. 382*

- Se ha construido un prototipo o incremento.

Hito técnico: Prueba Orientada a Objetos

- La corrección y compleción de análisis y del modelo de diseño han sido revisados.
- Se ha desarrollado y revisado una red de clases –responsabilidades- colaboraciones.

*Especificación de notación Orientada a Objetos UML*

El Lenguaje para Modelamiento Unificado (UML), es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Fue originalmente concebido por la Corporación Rational Software y tres de los más prominentes metodologistas en la industria de la tecnología y sistemas de información: Grady Booch, James Rumbaugh, e Ivar Jacobson. El lenguaje ha ganado un significante soporte de la industria de varias organizaciones vía el consorcio de socios de UML y ha sido presentado al Object Management Group (OMG) y aprobado por éste como un estándar (noviembre 17 de 1997).

UML se basa en las anteriores especificaciones BOOCH, RUMBAUGH y COAD-YOURDON. Divide cada proyecto en un número de diagramas que representan las diferentes vistas de proyecto. Estos diagramas juntos son los que representan la arquitectura del proyecto. A diferencia las especificaciones OO anteriores que presentaban diagramas que daban una impresión estática del sistema, UML introduce nuevos diagramas que representan una visión dinámica del sistema. Gracias al diseño de la parte dinámica del sistema, podemos darnos cuenta en la fase de diseño, de

problemas de la estructura que pueden propagar errores o de las partes que necesitan ser sincronizadas, así como del estado de cada una de las instancias en cada momento.

UML es ahora un estándar, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura o de cualquier otro ramo.

UML presenta el modelo OO en diagramas que usan la anotación pertinente y la suma de estos diagramas crean las diferentes vistas.

Las Vistas existentes en UML son:

- Vistas de Casos de Uso.
- Vistas de Diseño.
- Vistas de Procesos.
- Vistas de Implementación.
- Vistas de Despliegue.

Se dispone de dos tipos de diagramas, los que dan una vista estática del sistema y los que dan una visión dinámica.

Los diagramas estáticos son:

- ✓ Diagrama de Clases.
- ✓ Diagrama de Objetos.
- ✓ Diagrama de Componentes.
- ✓ Diagrama de Despliegue.
- ✓ Diagrama de Casos de Uso.

Los diagramas dinámicos son:

- ✓ Diagrama de Secuencia
- ✓ Diagrama de Estados.
- ✓ Diagrama de Actividades.

Los diagramas recomendados para usar en el análisis del sistema son:

- Diagramas de Casos de Uso.
- Diagramas de Actividad.
- Diagramas de Secuencia.
- Diagramas de Colaboración.
- Diagramas de Interfaces (Bosquejos).

Los diagramas recomendados para usar en el Diseño del Sistema son:

- Diagrama de Clases.
- Diagrama de Estados.
- Diagrama de Actividad.

Los diagramas recomendados para usar en la programación o codificación, e implementación del sistema son:

- Bases de Datos (Objeto-Relación).
- Entornos de Programación Visual.
- Diagramas de Distribución

**Análisis Orientado a Objetos**

El objetivo del análisis orientado a objetos es desarrollar una serie de modelos que describan el software de computadora al trabajar para satisfacer un conjunto de requisitos definidos por el cliente.

Para construir un modelo de análisis se aplican cinco principios básicos:

- Se modela el dominio de la información
- Se describe la función del módulo.
- Se representa el comportamiento del modelo.
- Los modelos se dividen para representar mas detalles.
- Los modelos Iniciales representan la esencia del problema mientras que los últimos aportan detalles de la implementación.

El propósito del análisis orientado a objetos es definir todas las clases (y las relaciones y comportamientos asociadas con ellas) que son relevantes al problema que se va a resolver. Para cumplirlo se deben ejecutar las siguientes tareas [8]:

[8] Ingeniería del Software. Roger S. Pressman. Pag. 390

1. Los requisitos básicos deben comunicarse entre el cliente y el ingeniero de software (Escenarios o **casos de uso y modelo de requisitos**).
2. **Identificar las clases** (es decir, definir atributos, operaciones para cada objeto del sistema y métodos).
3. Se debe especificar una **jerarquía y estructura** para organizar las clases.
4. Representar las relaciones objeto por objeto (**objeto-relación**).
5. Modelar el comportamiento del objeto (**objeto-comportamiento**)
6. Repetir iterativamente las tareas de la 1 a la 5 hasta completar el modelo.

*Casos de uso y modelado de Requisitos*

La recopilación de requisitos es el primer paso en el análisis del software. En éste el cliente y el ingeniero del software definen los requisitos básicos.

Creamos un conjunto de escenarios para identificar una parte del uso que se le dará al sistema. Esto se logra con los Diagramas de Casos de Uso (Figura 4). Sustituyen a los DFDs.

Primeramente debemos identificar los actores (persona o dispositivos externos) que usan el sistema. Debe entenderse como actores a la cualidad desempeñada como programador, operador, digitador, auditor, etc y no a la persona en particular, puede existir una sola persona que haga las cuatro cosas pero para el modelo serán cuatro actores.

Los actores pueden ser primarios (los que trabajan directamente con el software) y secundarios (los que dan soporte).

Cada caso de uso aporta un escenario no ambiguo de interacción entre un actor y el software. Pueden utilizarse para especificar requisitos de tiempo u otras restricciones para el escenario.

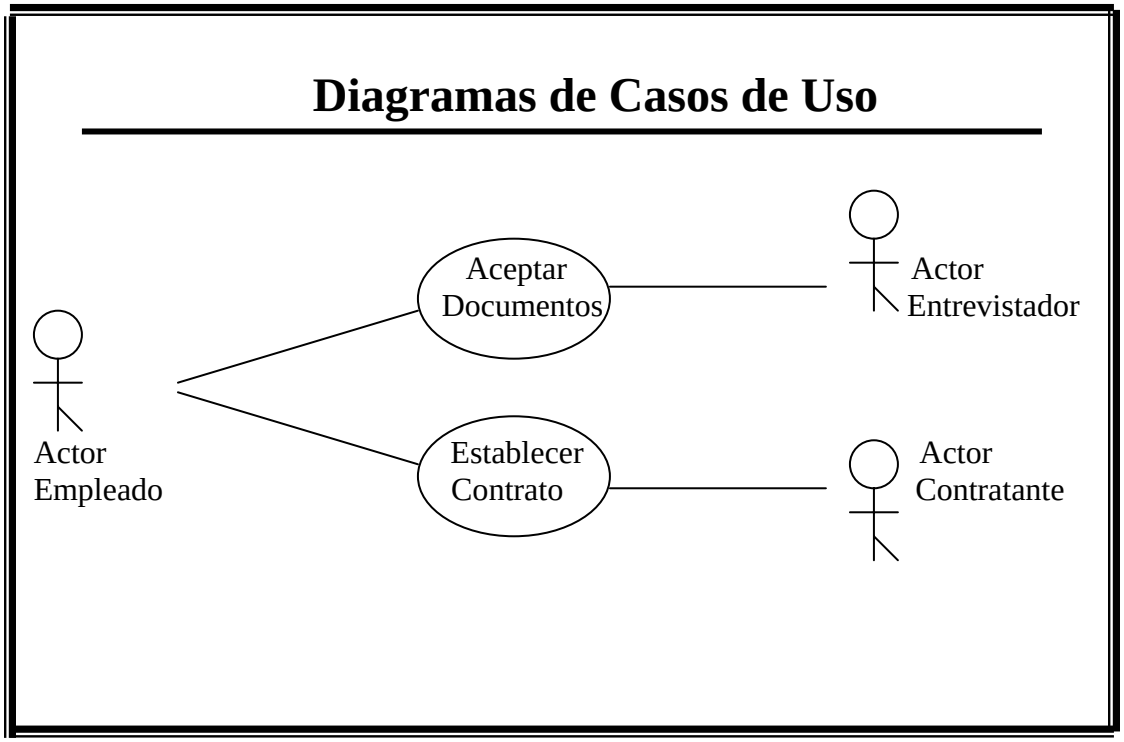



Figura 4

En la construcción de casos de uso se deben tener presente las siguientes pautas:

- Debe ser simple, intangible, claro y conciso.
- Generalmente hay pocos actores asociados a cada caso de uso.
- Preguntas claves a solventar:

- ¿Cuáles son las tareas del actor?
- ¿Qué información crea, guarda, modifica, destruye o lee el actor?
- ¿Debe el actor notificar al sistema los cambios externos?
- ¿Debe el sistema informar al actor de los cambios internos?

La descripción del caso de uso comprende:

- El inicio: cuándo y qué actor lo produce?
- El fin: cuando se produce y que valor devuelve?
- La interacción actor-caso de uso: que mensajes intercambian ambos?
- Objetivo del caso de uso: que lleva a cabo o intenta?
- Cronología y origen de las interacciones
- Repeticiones de comportamiento: que operaciones son iteradas?
- Situaciones opcionales: que ejecuciones alternativas presenta el caso de uso?

Seguidamente se debe usar un Diagrama de Actividades (Figura 5).

El diagrama de actividades es una variante de los diagramas de estado, organizado respecto de las acciones y principalmente destinado a representar el comportamiento interno de un método (la realización de una operación), de un caso de uso o de un proceso de negocio. Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina, se desencadena el paso de la siguiente actividad.

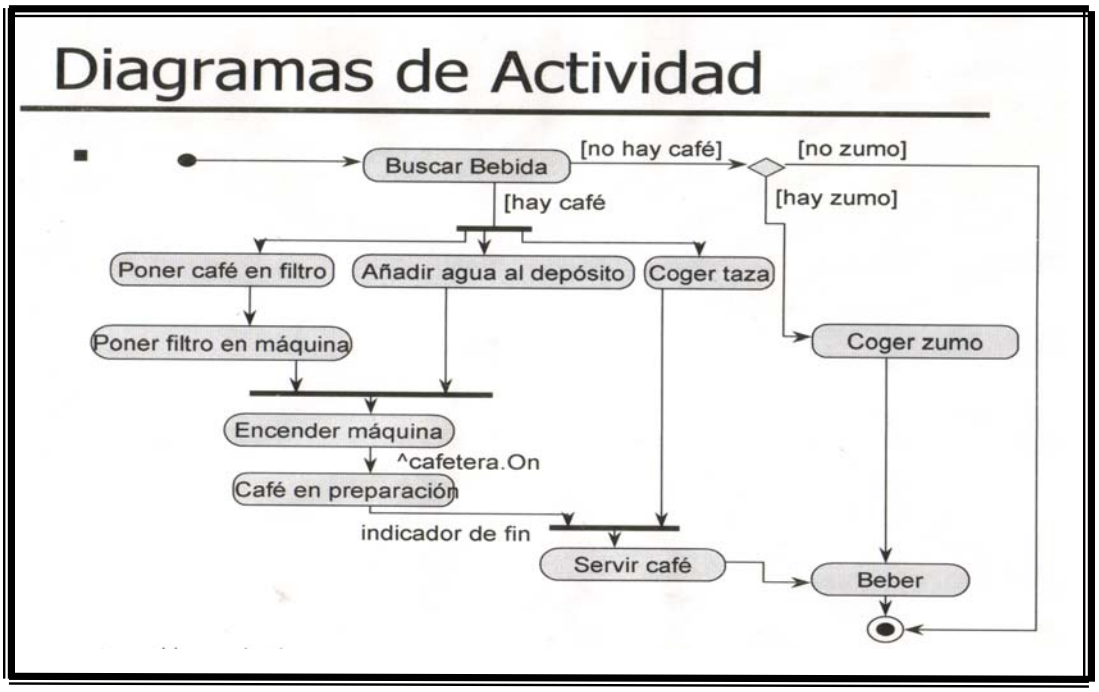



Figura No. 5

*Modelado de Clases-Responsabilidades- Colaboraciones (CRC)*

Después de tener los escenarios básicos, tenemos que identificar las clases e indicar sus responsabilidades (atributos y operaciones relevantes de la clase) y colaboraciones (solicitudes de información o solicitudes de alguna acción).

El modelo CRC es una colección de tarjetas índice estándar que representan clases. (Figura 6).

En la primera parte del modelado de objetos se determinó como identificar las clases, objetos y atributos. UML los encapsula a estos tres en paquetes.

Después de crear las clases utilizamos los diagramas de interacción (de Secuencia y de Colaboración). El diagrama de secuencia (Figura 7) que nos muestra todos los pasos del sistema o actividades relacionando procesos, flujos y entidades (gente o almacenamientos). Este muestra la secuencia de mensajes entre objetos durante un escenario concreto.

Cada objeto viene dado por una barra vertical y el tiempo transcurre de arriba abajo. Cuando existe demora entre el envío y la atención, se puede indicar con una línea oblicua.

| **MODELO CRC DE TRAJETA INDICE** | |
|---|---|
| Nombre de la Clase: | |
| Tipo de Clase: (dispositivo, propiedad, rol, evento,...) | |
| Características de la Clase: (tangible, atómica, concurrente,..) | |
| Responsabilidades:<br><br>(Atributos y Operaciones) | Colaboraciones:<br><br>(Solicitudes de cliente a servidor) |
| | |
| | |
| | |

Figura 6

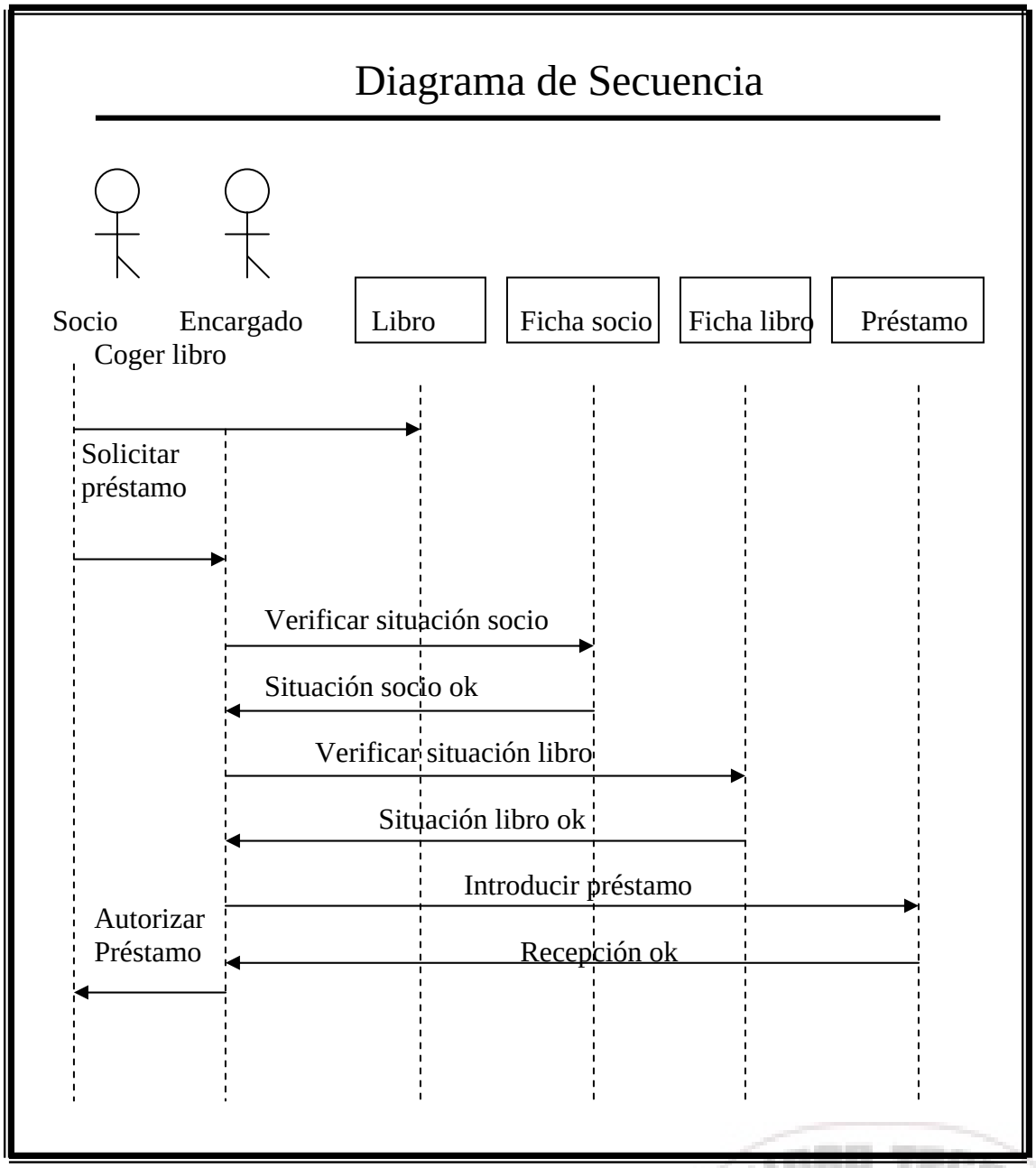



Figura 7

Después de los diagramas de secuencia, se crean los diagramas de Colaboración (Figura 8) que ofrecen una visión espacial mostrando los enlaces de comunicación

entre objetos. Los enlaces dan al sistema una presentación estática, los mensajes en estos enlaces presentan una visión dinámica.

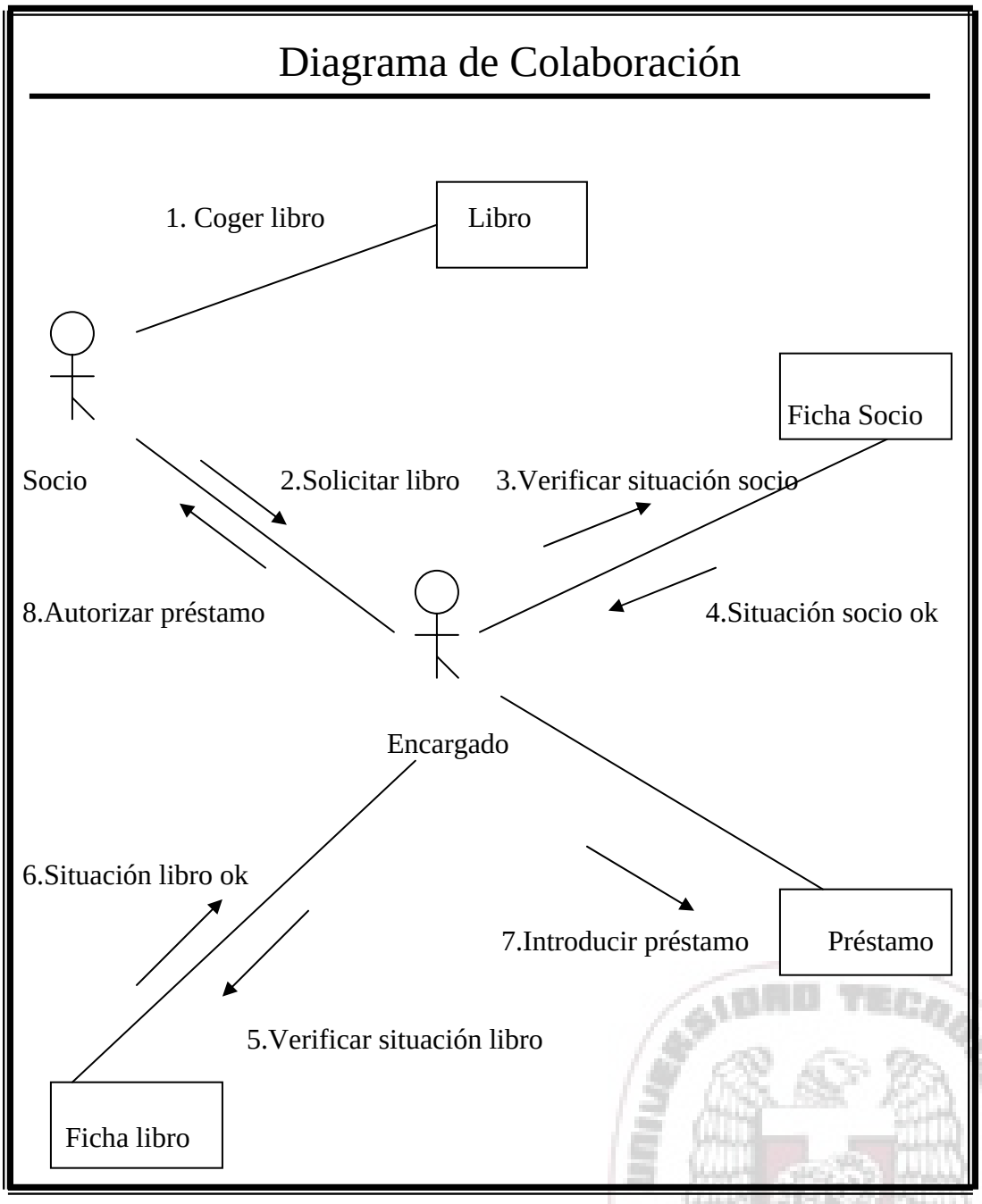



Figura 8

*Definición de Estructuras y Jerarquías*

Una vez que se han identificado las clases y los objetos usando el modelo CRC, el analista comienza a centrarse en la estructura de modelo de clases y las jerarquías resultantes que surgen al emerger clases y subclases.

El Diagrama de Clases nos permite ver las relaciones estructurales entre los objetos y las clases del sistema, así como su herencia.

Este diagrama se forma de la visualización que presentan los diagramas de secuencia y de colaboración.

Un ejemplo de una clase (Figura 9.1) y del Diagrama de Clases (Figura 9.2) es el siguiente:

| **Alumno** |
| --- |
| ♦ DNI: char [10]<br>♦ Número_exp: Int<br>♦ Nombre: char [50] |
| ➢ Alta()<br>➢ Poner_nota(asignatura: char *, año:int, nota:float)<br>➢ Matricula (Cursos:Asignatura, año:int)<br>➢ Listar-expediente() |

Figura 9.1

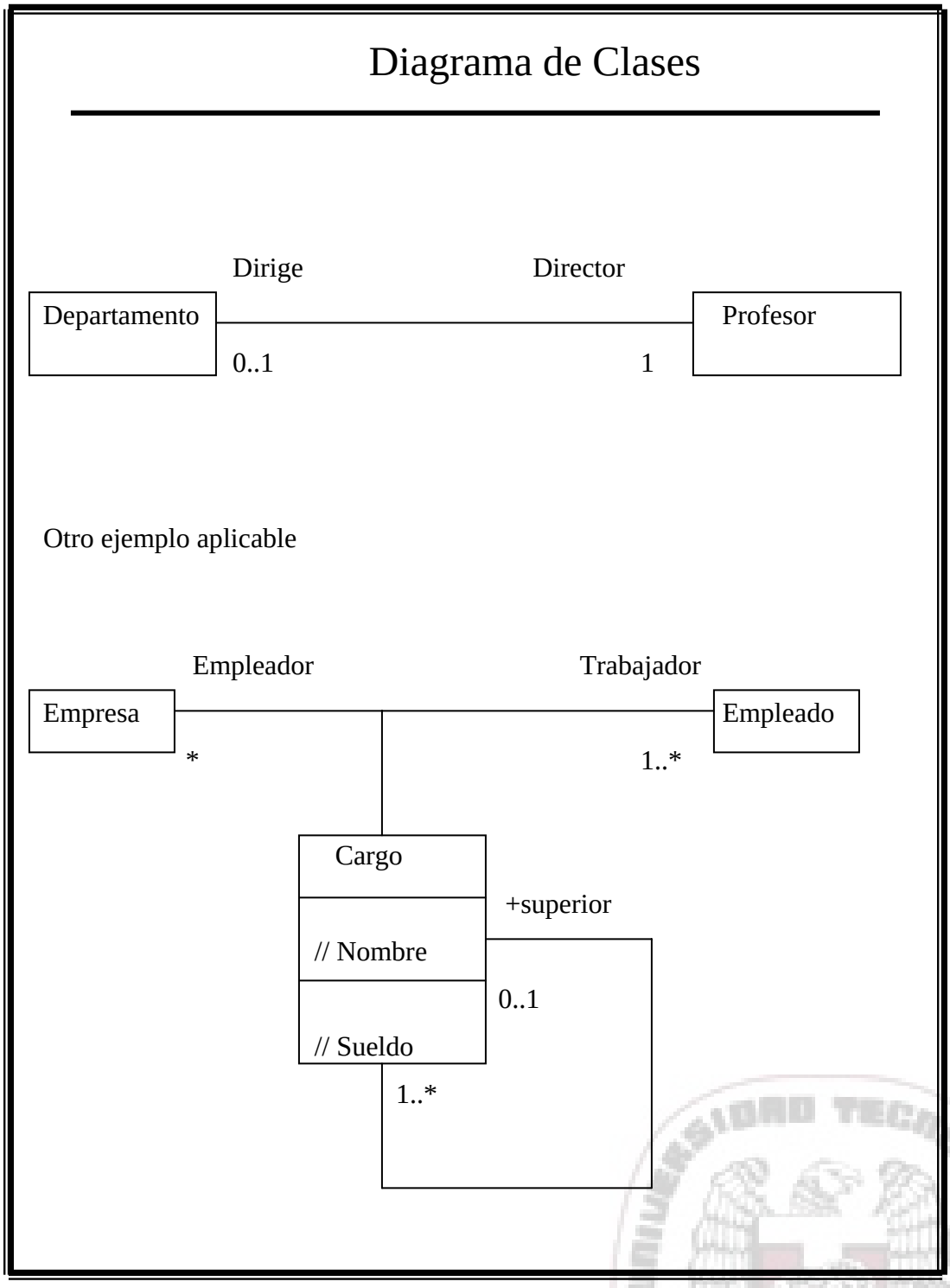



Figura 9.2

*Modelo Objeto-Relación en el Análisis*

El primer paso en el establecimiento de las relaciones es comprender las responsabilidades de cada clase. La tarjeta índice del modelo CRC contiene una lista de responsabilidades. El siguiente paso es definir aquellas clases colaboradoras que ayudan en la realización de cada responsabilidad. Esto establece la relación entre las clases.

Una relación existe entre dos clases cualesquiera que estén conectadas. Debido a esto los colaboradores siempre están relacionados de alguna manera. El tipo de relación mas común es la binaria. Las relaciones pueden derivarse a partir del examen de los verbos o frases verbales en el establecimiento del alcance o casos de uso para el problema. Usando un análisis gramatical, el analista aísla verbos que indican localizaciones físicas o emplazamientos (cerca de, parte de, contenido en), comunicaciones (transmite a, obtenido de), propiedad (incorporado por, se compone de) y cumplimiento de una condición (dirige, coordina, controla). Estos aportan una condición de relación.

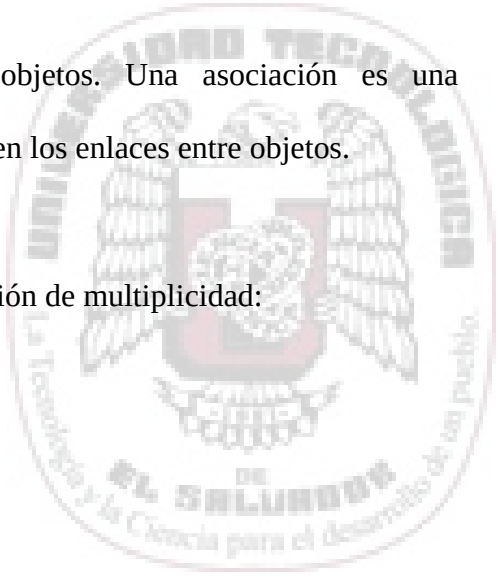
Formas de relación entre clases:

- Asociación y Agregación

  Expresa una conexión bidireccional entre objetos. Una asociación es una abstracción (Figura 10) de la relación existente en los enlaces entre objetos.

  Se usa la siguientes simbología para especificación de multiplicidad:

  1 Uno y solo uno



0..1 Cero o uno

M..N Desde M hasta N (enteros naturales)

* Cero o muchos

0..* Cero o muchos

1..* Uno o muchos (al menos uno)

La multiplicidad mínima >=1 establece una restricción de existencia.

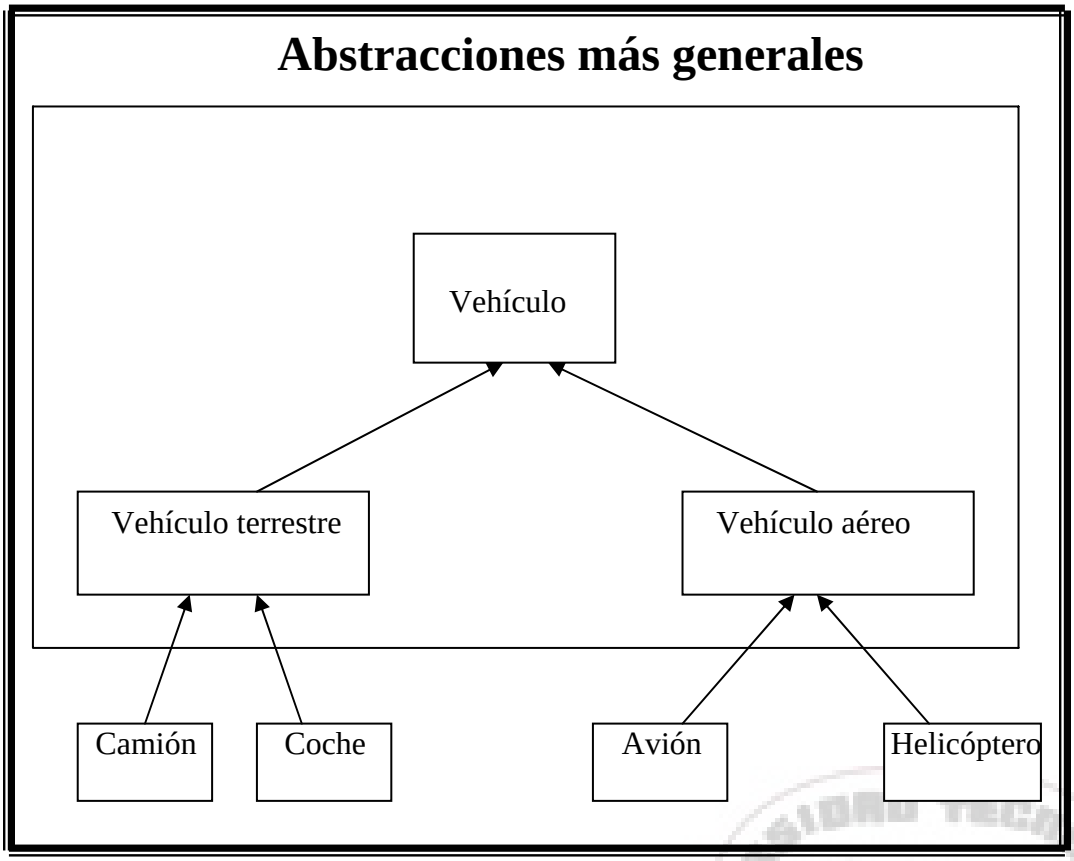



Figura 10

La agregación representa una relación parte_de entre objetos. Puede ser caracterizada con precisión determinando las relaciones de comportamiento y

estructura que existen entre el objeto agregado y cada uno de sus objetos componentes.

- Generalización/Especialización

Consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general. Un ejemplo es el siguiente:

*Modelo Objeto-Comportamiento en el Análisis*

El modelo CRC y el del objeto-relación representan elementos estáticos del modelo de análisis orientado a objetos. Ahora es el momento para hacer una transición al comportamiento dinámico del sistema. Para ejecutar este paso debemos representar el comportamiento del sistema como una función de eventos específicos y tiempo.

El modelo objeto-comportamiento indica como responderá un sistema OO a eventos externos o estímulos. Para crear el modelo deben crearse los Diagramas de Estado.

Los diagramas de estado (Figura 11) representan autómatas de estados finitos, desde el punto de vista de los estados y las transiciones. Son útiles solo para los objetos con un comportamiento significativo.

Se deben tomar en cuenta las siguientes bases para diseñar los diagramas:

- Cada objeto esta en un estado de cierto en un instante.

- El estado está caracterizado parcialmente por los valores de los atributos del objeto.
- El estado en el que se encuentra el objeto determina su comportamiento.
- Cada objeto sigue el comportamiento descrito en el diagrama de estado asociado a su clase.
- Los diagramas de estados y escenarios son complementarios.

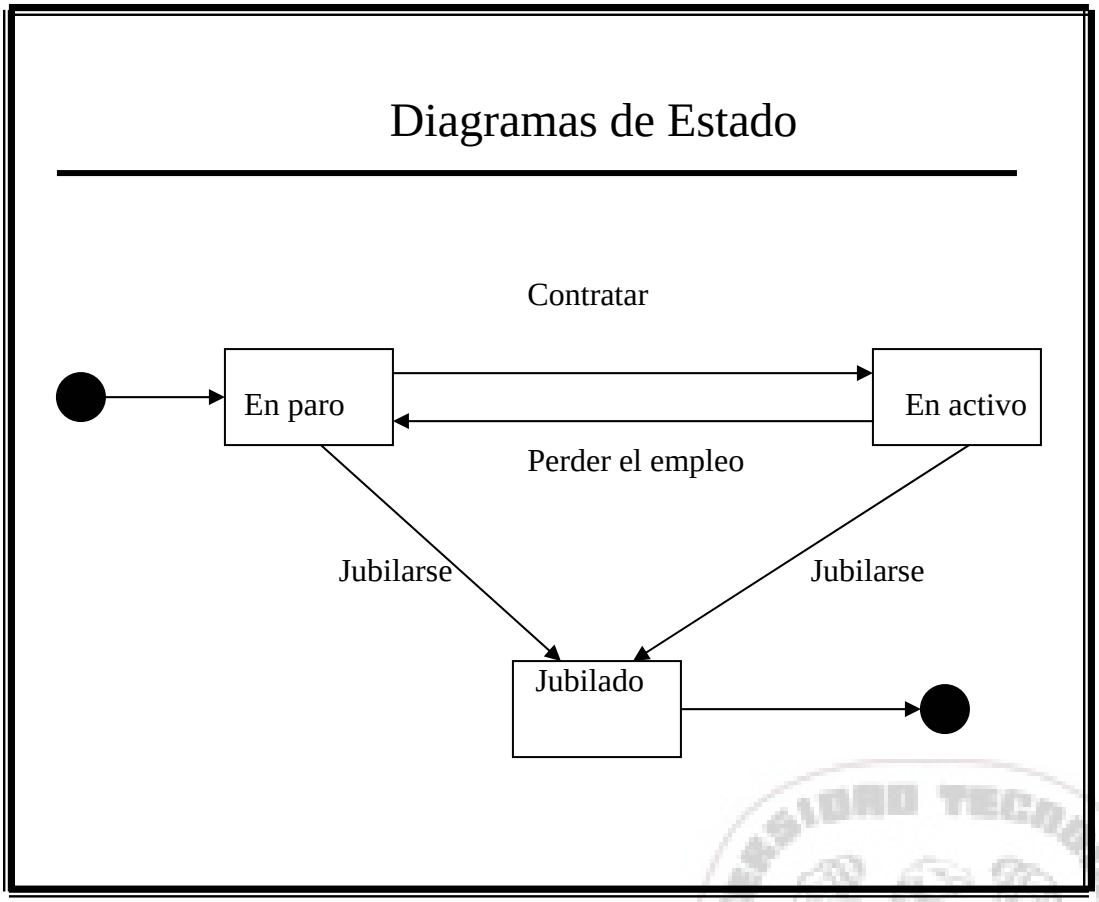



Figura 11

Con la representación grafica presentada en el análisis orientado a objetos podemos sentar las bases para la creación de una especificación de requisitos del software.

**Diseño Orientado a Objetos**

El diseño orientado a objetos (DOO) transforma el modelo de análisis orientado a objetos en un modelo de diseño que sirve como anteproyecto para la construcción del software. Este tipo de diseño logra cierto tipo de modularidad. Los componentes principales del sistema están organizados en módulos denominados subsistemas. Los datos y las operaciones que manipulan los datos están encapsulados en objetos, una forma modular que es el bloque de construcción de un sistema OO.

La naturaleza única del diseño orientado a objetos descansa en su capacidad de apoyarse en cuatro importantes conceptos de diseño del software: Abstracción, Ocultación de información, Dependencia funcional y Modularidad.

La figura 12 nos muestra las cuatro capas del diseño, estas son:

*La capa del subsistema:* Contiene una representación de cada uno de los subsistemas que le permiten al software conseguir que los requisitos definidos por el cliente se cumplan e implementar la infraestructura técnica que los soporta.

*La capa de clases y objetos:* Contiene las jerarquías de clases que permiten crear el sistema usando generalizaciones y especializaciones mejor definidas incrementalmente. Esta capa también contiene representaciones de diseño para cada objeto.

*La capa de mensajes:* Contiene los detalles que le permiten a cada objeto comunicarse con sus colaboradores. Esta capa establece las interfaces externas e internas para el sistema.

*La capa de responsabilidades:* Contiene las estructuras de datos y el diseño algorítmico para todos los atributos y operaciones de cada objeto.

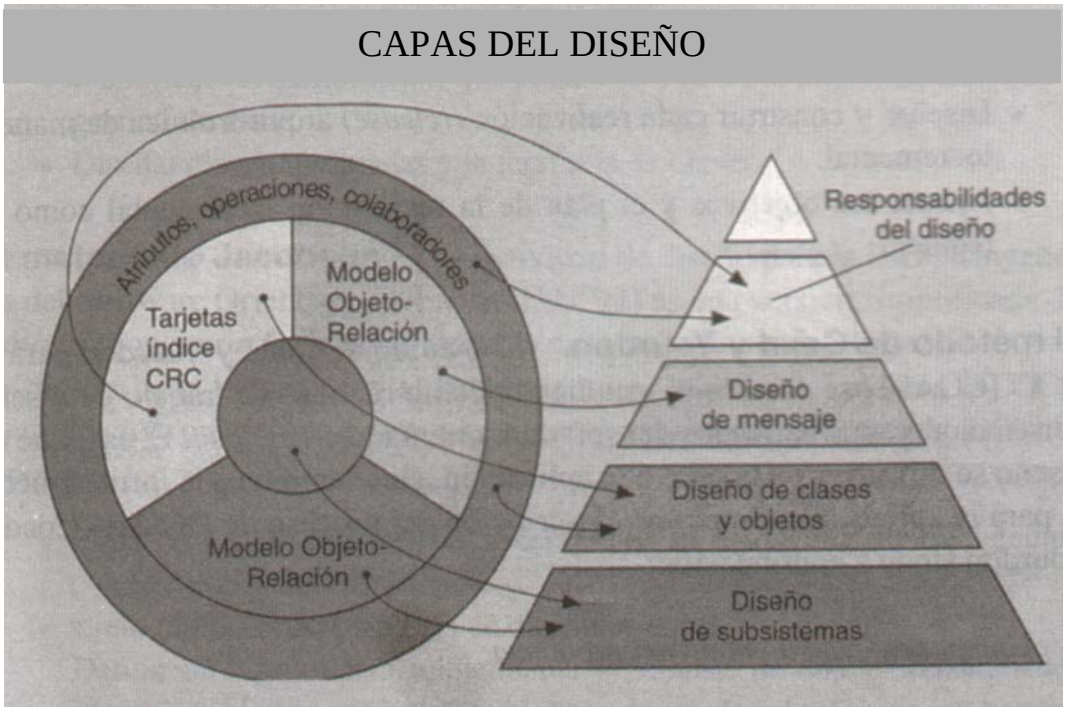



Figura 12

En la fase del diseño, hacemos uso de los Diagramas de Clases, los Diagramas de Estados y Diagramas de Actividad descritos anteriormente.

Durante el diseño de los subsistemas, es necesario para el ingeniero del software definir cuatro importantes componentes del diseño:

- ✓ Dominio del problema .
- ✓ Interacción humana.
- ✓ Gestión de tareas.
- ✓ Gestión de datos.

*Proceso de Diseño del Sistema*

El modelo de procesos para el diseño del sistema debe tener las siguientes etapas:

- Dividir el modelo de análisis en subsistemas.
- Identificar la concurrencia dictada por el problema .
- Asignar subsistemas a procesadores y tareas.
- Elegir una estrategia básica para la gestión de datos.
- Identificar los recursos globales y los mecanismos de control necesarios para acceder a ellos.
- Diseñar un mecanismo de control apropiado para el sistema.
- Considerar como manipular las condiciones límite.
- Revisar y considerar los intercambios.

*División del modelo de análisis en subsistemas*

En el diseño de sistemas orientado a objetos se divide el modelo de análisis para definir colecciones cohesivas de clases, relaciones y comportamientos. Estos elementos del diseño se empaquetan (PAQUETES) como un subsistema. Cada paquete puede identificarse por los servicios que realiza. Dicho servicio es una colección de operaciones que realizan una función específica (Ej. manejo de archivos de un

procesador de textos, producción de una representación tridimensional, convirtiendo una señal de video análoga en señal digital).

El flujo de comunicación e información que implican los enlaces entre los subsistemas pueden representarse en DFDS, donde cada burbuja es un subsistema.

Los diagramas de colaboración nos permiten definir las colaboraciones entre los subsistemas cliente/servidor.

*Identificación de concurrencias y asignación de subsistemas a procesadores*

A partir del modelo objeto-comportamiento encontramos la concurrencia entre objetos o subsistemas. Específicamente, a partir de los diagramas de estado. Cuando los subsistemas no están activos al mismo tiempo, pueden implementarse dentro de un mismo hardware procesador y no hay necesidad de procesamiento concurrente; por otra parte cuando estos actúan sobre eventos asíncronamente y al mismo tiempo, se toman como concurrentes. Cuando los subsistemas son concurrentes, existen dos opciones de asignación:

- ✓ Asignar cada subsistema a un procesador independiente
- ✓ Asignar los subsistemas al mismo procesador y ofrecer soporte de concurrencia a través de las capacidades del sistema operativo.

Si al examinar el diagrama de estados, indica que solo existe un único objeto activo en un momento, se establece un hilo de control, así cuando un subsistema esta activo en

diferentes tareas se establecen hilos por separado y cada una está definida como una tarea independiente.

*Estrategia para la gestión de datos*

La gestión de datos abarca dos áreas distintas: 1) la gestión de datos críticos para la propia aplicación, y 2) la creación de una infraestructura para el almacenamiento y recuperación de datos. Esta gestión se diseña por capas.

El diseño de la componente para la gestión de datos incluye el diseño de los atributos y operaciones necesarias para la gestión de los datos. Se sugiere una clase llamada servidor-de-objetos con servicios de informar a cada objeto que almacene o recupere objetos almacenados para ser usados por otros componentes del diseño.

En este gestionar de datos es necesario hacer un Diccionario de Datos que describa cada tabla y consecuentemente mostrar el diseño de una base de datos relacional.

*Identificación de recursos y mecanismos de control*

Los recursos globales del sistema pueden ser entidades externas (una torre de discos, procesador, o línea de comunicación) o abstracciones (una base de datos, un objeto). El ingeniero de software debe diseñar un mecanismo de control sobre él.

*El componente de interfaz  hombre-maquina (IHM)*

El modelo de análisis orientado a objetos contiene escenarios de uso (llamados casos de uso) y una descripción del papel que tienen los usuarios (llamados actores) al interactuar con el sistema. Esto sirve como dato de entrada al proceso de diseño IHM.

Una vez que se han definido el actor y su escenario de uso, se identifica una jerarquía de órdenes, la jerarquía de comandos define las principales categorías de menús del sistema (las ventanas de menús). La jerarquía de comandos se refina iterativamente hasta que cada caso de uso pueda implementarse navegando a través de la jerarquía de funciones.

El implementador solo necesita identificar los objetos que poseen las características para el dominio del problema para construir con iconos, ventanas, operaciones de ratón y una amplia variedad de funciones de interacción.

Una variedad de representaciones incluidas en el modelo de análisis y el diseño del sistema aportan una especificación de todas las operaciones y atributos. Los algoritmos y estructuras de datos se diseñan usando un enfoque que difiere poco del enfoque del diseño de datos y los procedimientos aplicados a ingeniería de software convencional.

Para diseñar parte de la implementación se utilizan los Diagramas de Distribución (relación entre el servidor central y sus elementos) y Diagramas de

Componentes(muestra la relación entre los diferentes componentes como el hardware, software, DBF, conexiones, etc).

**Pruebas Orientadas a Objetos**

El objetivo de la realización de pruebas es encontrar el mayor número posible de errores con una cantidad de esfuerzo racional a lo largo de un espacio de tiempo realista. Aunque este objetivo fundamental permanece el mismo tiempo para el software orientado a objetos, la naturaleza de los programas OO cambia la estrategia y las tácticas de la prueba.

Para probar adecuadamente los sistemas OO, deben hacerse tres cosas: 1) la definición de la prueba puede ampliarse para incluir técnicas de detección de errores aplicados a los modelos DOO y AOO; 2) la estrategia para la unidad e integración deben cambiar significativamente; 3) el diseño de casos de prueba debe tener en cuenta las características propias del software orientado a objetos.

Durante el análisis podemos eliminar atributos que se han diseñado por error, de esa manera podemos evitar esfuerzos innecesarios como los siguientes:

- ✓ Se puede haber generado subclases especiales para acomodar el atributo innecesario o sus excepciones. Se evita el trabajo involucrado en la creación de las clases innecesarias.
- ✓ Una mala interpretación de la definición de la clase puede conducir a relaciones de clases incorrectas o innecesarias.

- ✓ El comportamiento del sistema o sus clases puede ser impropiamente caracterizado para acomodar el atributo extraño.

Durante el diseño puede repercutir el problema de la siguiente manera:

- ✓ Puede ocurrir una asignación impropia de clases a subsistemas y/o tareas durante el diseño del sistema.
- ✓ Se realizaría un esfuerzo de trabajo no necesario para crear el diseño procedimental de las operaciones relacionadas con el atributo innecesario.
- ✓ El modelo de intercambio de mensajes sería incorrecto (puesto que los mensajes deben diseñarse para las operaciones que son extrañas).

Durante el análisis y diseño, se debe juzgar la corrección semántica basándose en la conformidad del modelo con el dominio del problema del mundo real. De cumplirse esto, entonces está semánticamente correcto. Para lograrlo se debe revisar las definiciones de clases y jerarquías en busca de omisiones y ambigüedades; además deben examinarse las relaciones de clases (conexiones de instancias).

La consistencia de los modelos de AOO y DOO pueden juzgarse a través de una consideración de las relaciones entre entidades en el modelo. Un modelo inconsistente tiene representaciones que no son correctamente reflejadas en otras partes del modelo. Para valorar la consistencia, deberán examinarse cada clase y sus conexiones a otras clases. Puede usarse el modelo de colaboración y un diagrama objeto-relación.

*Diseño de casos de prueba para software Orientado a Objetos*

Entre los métodos de casos de prueba OO se sugiere el siguiente:

- Cada paso de prueba debe estar identificado unívocamente y asociado explícitamente con la clase a probar.
- Debe establecerse el propósito de la prueba.
- Desarrollar una lista de pasos de prueba para cada prueba, la cuál deberá contener:
  - Una lista de estados especificados para el objeto a probar.
  - Una lista de mensajes y operaciones a ejecutar como consecuencia de la prueba.
  - Una lista de excepciones que pueden ocurrir al probar el objeto.
  - Una lista de condiciones externas (cambios en el entorno externo al software que deben existir para conducir propiamente la ejecución de la prueba).
  - Información suplementaria que ayudará en la comprensión o implementación de la prueba.

Con las etapas de Análisis, Diseño y Pruebas tenemos un modelado apropiado para poder construir la aplicación con un lenguaje de programación orientado a objetos. El siguiente apartado presenta la teoría para éste propósito.

### 1.3.2 Sistema Gestor de Base de Datos Relacionales

**Diseño de la Base de Datos de la Aplicación**

El diseño de la base de datos se concretiza con un Diccionario de Datos y las especificaciones de las relaciones entre las tablas.

El diccionario de datos es un catálogo, un depósito, de los elementos en un sistema. Como su nombre lo sugiere, estos elementos se centran alrededor de los datos y la forma en que están estructurados para satisfacer los requerimientos de los usuarios y las necesidades de la organización. En este diccionario se encuentra la identificación del almacén de datos, las estructuras de datos y el elemento dato.

El diccionario de datos se utiliza por cinco razones:

- Para manejar los detalles en sistemas grandes.
- Para comunicar un significado común para todos los elementos del sistema.
- Para documentar las características del sistema.
- Para facilitar el análisis de los detalles con la finalidad de evaluar las características y determinar donde efectuar cambios en el sistema.
- Localizar errores y omisiones en el sistema.

*Almacén de Datos*

Las tablas de datos son clasificadas según su función, entre ellas están los archivos maestros, los de transacciones y los catálogos. Debe especificarse claramente el nombre de cada tabla y su propósito tal como se muestra en la figura 13.1.

*Estructuras de Datos*

Es un grupo de datos elementales que están relacionados con otros y que en conjunto describen un componente del sistema. Por ejemplo la estructura de datos de censo de empleados, esta definida por un grupo de datos elementales que incluyen el código del

empleado, edad, sexo, dirección particular y otros detalles que describen la entidad en estudio. Las estructuras de los datos las podemos describir como se muestra en la figura 13.2.

*Elemento Dato*

Este es el nivel mas importante de los datos. Tiene mucha importancia para los analistas de sistemas. El elemento datos hace referencia a código de empleado, edad, sexo, etc. El formato para plasmar esta información se muestra en la figura 14.

| **LISTA DE ALMACENAMIENTOS DE DATOS** | |
|---|---|
| **Tabla No.** | **Nombre de la Tabla** |
| 1 | Maestro de Empleados |
| 2 | Movimientos de Permisos sin goce de sueldo |
| n. | Nombre de tabla n |

Figura 13.1

| **ESTRUCTURAS DE DATOS** |
|---|
| Nombre de Almacén de Datos: |
| Descripción: |
| Descripción de Datos: (Lista de campos o atributos) |
| |
| Volumen: (registros en un período, % crecimiento etc.) |
| Acceso: (secuencial, indexado, etc.) |

Figura 13.2

| ELEMENTO DATO |
| --- |
| Nombre del Elemento Dato: |
| Descripción: |
| |
| Tipo: |
| Longitud: |
| Alias: |
| Rango de Valores: |
| Lista de Valores específicos: (si existen) |
| Otros detalles de edición: |

Figura 14

**Relaciones entre tablas y datos**

Los analistas de sistemas deben ser capaz de especificar las relaciones entre entidades y tablas. El software de diseño de la base de datos puede proporcionar la herramienta que permita hacer el diagrama *entidad-relación*. Normalmente se presentan las tablas en forma de rectángulo y las relaciones entre ellas se describen mediante su dependencia una de la otra, al igual que por el enlace de la relación.

Existen dos tipos de dependencia entre entidades. En la primera, la *dependencia existencial*, una entidad no puede existir a menos que la otra esté presente; el que exista la segunda depende de la primera. Por ejemplo, un registro de incapacidad de salud no puede existir si no hay un empleado a quien se le aplique.

En el otro tipo de dependencia, la *dependencia de identificación*, una entidad no puede identificarse de manera única con sus propios atributos. La identificación es posible sólo mediante las relaciones de una entidad con otras. Para identificar una entidad se deben conocer las otras. Por ejemplo, cargos de los empleados son únicos dentro de los departamento, pero debemos conocer el departamento para identificar que esta entidad es una actividad.

Las asociaciones entre entidades o tablas existen de uno a muchos (uno o mas apariciones de la entidad correspondiente) o uno a uno (existe una y solo una aparición correspondiente de la otra entidad en la relación).

**Normalización**

La normalización es el proceso de simplificar la relación entre los campos del registro. Por medio de la normalización, un conjunto de datos en un registro se reemplaza por varios registros que son mas simples y predecibles y, por lo tanto, mas manejables. La normalización se lleva a cabo por cuatro razones:

- Estructurar los datos en forma que se puedan representar las relaciones pertinentes entre los datos.
- Permitir la recuperación sencilla de los datos en respuesta a las solicitudes de consultas y reportes.
- Simplificar el mantenimiento de los datos actualizándolos, insertándolos y borrándolos.

- Reducir la necesidad de reestructurar o reorganizar los datos cuando surjan nuevas aplicaciones.

Las relaciones de entidades y la buena normalización de las tablas, dependerá en buena parte del software que se utilice para diseñar la base de datos de la aplicación.

**Software Administrador de Base de Datos Microsoft Access**

Microsoft Access es un Sistema Gestor de Bases de Datos Relacionales (SGBD) y entre las partes sobresalientes de éste podemos mencionar las siguientes:

*Tablas: S*on el componente básico o elemental de las bases de datos. O lo que es igual, una base de datos está principalmente compuesta por varias tablas relacionadas. Las tablas contienen datos sobre algo o alguien, proveedores, clientes, libros en una biblioteca, compras, ventas, etc. Un registro es un conjunto de información acerca de una persona, cosa o evento. Cada registro de una tabla contiene el mismo conjunto de campos y cada campo contiene el mismo tipo de información para cada registro. Los valores de los campos se usan para indicar los registros que se desean ver.

*Consultas: S*on preguntas que un usuario hace a la base de datos. Con ellas puede obtener información de varias tablas y con la estructura que más le interese; Además, las consultas pueden archivarse de forma que la próxima vez que se quiera hacer la misma pregunta no tendrá que volver a plantearla, será suficiente con llamar a la consulta previamente creada. La importancia de las consultas es enorme, de hecho es la

potencia de esta herramienta, la que permite que los gestores de base de datos sean casi imprescindibles en nuestro trabajo diario.

*Formularios: S*on un mecanismo que facilita enormemente la operatoria general con tablas, principalmente a la hora de mostrar, introducir y modificar datos. Un uso adecuado de éstos redunda bastante en el nivel de manejabilidad de una aplicación o de un sistema de información desarrollado con Access.

*Informes:* Permiten presentar la información con una apariencia altamente profesional a la hora de imprimir nuestros datos.

Los elementos de un formulario que permiten presentar o imprimir los datos se llaman controles. Con un control se pueden presentar los datos de un campo, los resultados de un cálculo, el texto para el título o para un mensaje, o bien un gráfico, una imagen u otro objeto, incluso otro formulario u otro informe.

*Páginas de acceso a datos:* Es una página Web que se puede utilizar para agregar, modificar, ver o manipular datos actuales en una base de datos de Microsoft Access o de SQL Server. Se pueden crear páginas que se utilizarán para especificar y modificar datos, de manera similar a los formularios de Access.

También se pueden crear páginas que muestren registros agrupados jerárquicamente, de manera similar a los informes de Access.

*Macros:* Son un mecanismo de automatización de Microsoft Access. Utilizando éstas es posible automatizar tareas repetitivas eliminando la posibilidad de introducir errores de operación y liberando tiempo para emplearlo en otras actividades. Podemos decir que una macro no es más que una lista de tareas que queremos que Access lleve a cabo automáticamente.

*Módulos:* Son objetos donde se almacena código escrito en lenguaje de programación denominado Access Basic.

| ALGUNOS TIPOS DE DATOS A UTILIZAR EN ACCESS | | |
|---|---|---|
| **VALOR** | **TIPO DE DATOS** | **TAMAÑO** |
| Texto | (Predeterminado). Se utiliza para almacenar texto o combinaciones de texto y números, así como números que no requieran cálculos, como los números de teléfono. | Hasta 255 caracteres |
| Numérico | Almacena datos numéricos utilizados en cálculos matemáticos. | 1, 2, 4 u 8 bytes. |
| Fecha/Hora | Almacena valores de fecha y hora para los años del 100 al 9999 | 8 bytes |
| Moneda | Guarda valores de moneda y datos numéricos utilizados en cálculos matemáticos en los que estén implicados datos que contengan entre uno a cuatro decimales. La precisión es de hasta 15 dígitos a la izquierda del separador decimal hasta 4 dígitos a la derecha del mismo. | 8 bytes |

Figura 15

| PROPIEDADES DEL CAMPO | |
|---|---|
| **Tamaño del campo** | Esta propiedad representa el máximo número de caracteres que puede ser introducido en un campo. Este valor puede variar dependiendo del tipo de dato que será introducido, por ejemplo:<br>Si es **CARACTER** varía entre : 1 hasta 255 caracteres<br>Si es **NUMERICO** varía entre : Entero, Entero Largo, Simple, etc. |
| **Formato del campo** | Determina la forma en que los números, las fechas, las horas, el texto o cualquier dato introducido se muestran e imprimen.<br>Esta propiedad utiliza diferentes valores dependiendo del tipo de dato del campo por ejemplo:<br>Si es de tipo Texto:<br>> Convierte todos los caracteres a mayúsculas<br>< convierte todos los caracteres a minúsculas<br>***[COLOR]:*** Asigna un color para los datos introducidos. Los colores disponibles son: negro, azul, verde, aguamarina, rojo, fucsia, amarillo y blanco. |
| **Máscara de entrada** | Es un formato que se especifica para que el usuario identifique como introducir los datos en un campo, por ejemplo cuando se introduce un número telefónico sabemos que se escribe 999-9999 .<br>Para definir una máscara de entrada puede auxiliarse de los siguientes caracteres:<br>L Letra (A a Z, el ingreso de datos es obligatorio)<br>0 Digito (0 a 9, el ingreso de datos es obligatorio, y los signos más [+] y menos [-] no son permitidos).<br>- Solo se muestra para separar caracteres de texto o Número. Este caracter no se guarda en el campo. |
| **Título del campo** | Representa una etiqueta que se mostrará en un formulario o cuando se abre la tabla. Si no se escribe la etiqueta, Access tomara como etiqueta el nombre del campo. |
| **Valor predeterminado** | Es un valor que se introduce automáticamente en un campo cuando se crea un nuevo registro, por ejemplo, si se escribe 0 (cero), entonces todos los nuevos registros aparecerán con el campo especificado lleno con 0. Otro ejemplo es, si tenemos un campo llamado sexo y se determina que todos los nuevos registros generados aparezcan con una M que indique que el sexo es masculino. |
| **Regla de validación** | Es una expresión que se introduce para determinar que tipos de valores pueden introducirse en un campo. |
| **Texto de validación** | Es un mensaje que aparecerá cuando el usuario intente ingresar un dato prohibido por la regla de validación. |
| **Requerido** | Permite determinar si el campo es necesario llenarlo o no.<br>Si es requerido está configurado como SI, entonces este campo siempre tendrá que llenarse y nunca será omitido, no obstante si requerido dice No podría omitir dicho campo. |
| **Indexado** | Indica si los datos almacenados estarán ordenados en forma alfabética, con el objeto de hacer la búsqueda de datos más rápida.<br>Esta propiedad puede tener las siguientes variantes:<br>No (Predeterminado) Sin índice.<br>Si (Con duplicados) El índice admite valores duplicados<br>Si (Sin duplicados) El índice no admite valores duplicados |

Figura 16

Access utiliza tres tipos principales de controles (figura 17), estos pueden ser un simple cuadro de texto o un cuadro de lista que ofrece varias opciones.

| CONTROLES DE ACCESS | |
|---|---|
| **TIPO** | **DESCRIPCIÓN** |
| Dependiente (Bound) | Un control dependiente (bound) es el que muestra información de un campo de una tabla o consulta (query) existente. Le permite mostrar, ingresar y actualizar los valores de su Base de Datos. |
| Independiente (Unbound) | Un control independiente (unbound) contiene información que no está vinculada a la base de datos. Por ejemplo, puede utilizar un control independiente para mostrar el logo de una compañía. También se puede usar controles independientes para almacenar datos en memoria en forma temporal. |
| Calculado (Calculation) | Un control calculado muestra un valor basado en una expresión que se creó. La expresión puede hacer referencia a un campo de la tabla o consulta subyacente al formulario; o a otro control en el formulario. |

Figura 17

Hasta ahora era un desafío reunir en un solo lugar los datos almacenados en varios archivos y formatos de archivo, como por ejemplo archivos de hojas de cálculo, archivos de texto y archivos de otras bases de datos. Microsoft Access proporciona dos métodos útiles para usar la información almacenada fuera de su base de Datos de Access.

- **Adjuntar una tabla externa.** Se puede usar Microsoft Access para ver y actualizar la información de otra base de datos de Microsoft Access, pero también la información guardada en un formato de archivo de base de datos diferente, como por ejemplo dBASE, Paradox, Btrieve o SQL Server, sin necesidad de copiar o

mover los datos de su origen. Para ello, debe adjuntar la tabla externa a la base de datos de Microsoft Access. Microsoft Access usa iconos especiales para las tablas externas en la ventana de la base de datos.

- **Importar datos.** También se puede importar a una tabla de Microsoft Access datos procedentes de una amplia gama de formatos de archivos, incluyendo archivos de hojas de cálculo, archivos de texto y archivos de otras bases de datos. Al importar información, se copian los datos desde su origen hasta una nueva tabla de Microsoft Access.

- **Combinar tablas.** Si Microsoft Access no ha combinado las tablas automáticamente o si no se ha creado ninguna relación entre las tablas, estas no aparecerán conectadas mediante líneas de combinación en la ventana Consulta. Aún así, se puede usar datos relacionados de las dos tablas si éstas se combinan de forma manual en la ventana .

Antes de combinar dos tablas en una consulta, los campos relacionados deben estar presentes en ambas tablas. La combinación funciona si los dos campos contienen datos que coinciden en los registros relacionados. Por ejemplo, si combina el campo ID de categoría de la tabla c*ategorías* con un campo de la tabla *productos* que no sea el campo ID de categoría, Microsoft Access no podrá relacionar los registros de las dos tablas, ya que los datos de los campos combinados no coinciden.

### 1.3.3 Desarrollo de la Aplicación con Visual Basic

Visual Basic es un ambiente gráfico de desarrollo de aplicaciones para el sistema operativo Microsoft Windows. Las aplicaciones creadas con Visual Basic están basadas en objetos y son manejadas por eventos. Cada formulario (ventana), menú o control que se crea con Visual Basic es un módulo autocontenido llamado objeto.

Visual Basic se deriva del lenguaje Basic, el cual es un lenguaje de programación estructurado. Sin embargo, Visual Basic emplea un modelo de programación manejada por eventos.

En las aplicaciones manejadas por eventos, la ejecución no sigue una ruta predefinida. En vez de esto, se ejecutan diferentes secciones de código en respuesta a eventos. Los eventos se desencadenan por acciones del usuario, por mensajes del sistema o de otras aplicaciones.

La secuencia de eventos determina el orden en que el código se ejecuta. Es por esto que la ruta que sigue el código de la aplicación es diferente cada vez que se ejecuta el programa.

**Los Formularios y Controles en Visual Basic**

Los formularios son objetos que exponen las propiedades que definen su apariencia, los métodos que definen su comportamiento y los eventos que definen la forma en que interactúan con el usuario. Mediante el establecimiento de las propiedades del

formulario y la escritura de código de Visual Basic para responder a sus eventos se personaliza el objeto para cubrir las necesidades de la aplicación.

Los *controles* son objetos que están contenidos en los formularios. Cada tipo de control tiene su propio conjunto de propiedades, métodos y eventos, que lo hacen adecuado para una finalidad determinada. Algunos de los controles que se pueden usar en las aplicaciones son más adecuados para escribir o mostrar texto, mientras que otros controles permiten tener acceso a otras aplicaciones y procesar los datos como si la aplicación remota formara parte del código.

Los bloques básicos de construcción de una aplicación con Visual Basic son los objetos. Cada objeto tiene un conjunto de características y un comportamiento definido (propiedades, métodos y eventos) que lo diferencian de otros tipos de objeto. En otras palabras, un objeto formulario ha sido diseñado para cumplir determinada función en una aplicación.

**El proyecto**

Una parte esencial de la programación manejada por eventos es el escribir código que responda a los posibles eventos que pueden ocurrir en una aplicación. Cuando desarrolla una aplicación, Visual Basic crea un archivo especial llamado Archivo de Proyecto para administrar todos los demás archivos de la aplicación. Esta información se actualiza cada vez que se guarda el proyecto. Todos los archivos y objetos también

se pueden compartir con otros proyectos. Un proyecto está compuesto por los siguientes tipos de archivos:

| TIPO DE ARCHIVO | EXTÉNSIÓN | DESCRIPCIÓN |
|---|---|---|
| Proyecto | .vbp | Realiza el seguimiento de todos los componentes de la aplicación. |
| Formulario | .frm, frx | Incluye el formulario, los objetos sobre el formulario y el código que se ejecuta cuando ocurre un evento en el formulario. |
| Módulo estándar | .bas | Contiene procedimientos **Sub** y **Function** que pueden ser invocados por cualquier formulario u objeto sobre el formulario (opcional). |
| Controles personalizados | .ocx | Controles adicionales a los controles estándar proporcionados por Microsoft u otras empresas. (opcional) |
| Módulo de clase | .cls | Contiene información binaria usada por la aplicación. Son usados generalmente cuando se crean programas para múltiples lenguajes. (opcional) |

Figura 18

**El Entorno Integrado de Desarrollo (IDE)**

Cuando se inicia Visual Basic, se crea un proyecto nuevo con un formulario. El IDE de Visual Basic consta de los siguientes elementos:

- Barra de Menús
- Barra de Herramientas
- Cuadro de Herramientas
- Diseñador de Formularios
- Explorador de Proyectos
- Ventana de Propiedades
- Ventana de Código

**Algunas de las características del lenguaje**

Se adhiere como cliente de diferentes administradores de base de datos compatibles como es el caso de Access, DBMS sencillo de pegarlo a Visual Basic como la base de datos a utilizar. Esta característica lo convierte en una opción para ser utilizado ya que puede trabajar simultáneamente en una base de datos creada en otro DBMS ya existente y permitiendo la migración en forma gradual.

Debido a que Visual Basic es un lenguaje orientado a objetos, es fácil pasar del diseño orientado a objetos, a la construcción del software.

Por otra parte Visual Basic proporciona una ambiente de interfaz gráfica que permite al usuario amigabilidad al interactuar con éste.

**Declaración de Variables**

**Dim| Public| Static** nombre_variable **As** tipo

**Dim:** Al declarar una variable con dim estamos diciendo que la variable sea local al ámbito en que se declara. Puede ser dentro de un procedimiento o dentro de un formulario, de esta forma no sería accesible desde los demás procedimientos o formularios.

**Public:** Las variables declaradas serán públicas y podrán estar accesibles desde todos los formularios de la aplicación. Para conseguirlo tendremos que declararlas en un módulo de código, no en la sección *declarations* de cualquier formulario de los que

conste la aplicación. Para crear un módulo de código en el menú principal de Visual Basic marcamos en INSERT/MODULE y aparecerá junto a los demás formularios de la ventana de proyecto aunque con un icono distinto indicando que se trata de un módulo de código.

**Static**: Con esta forma de declarar variables conseguiremos que las variables locales no se creen y se destruyan al entrar y salir de los procedimientos donde fueron declaradas sino que se mantenga su valor durante todo el periodo de ejecución de la aplicación. De esta forma al entrar en algún procedimiento las variables recuerdan el valor que tenían cuando se salió de él.

| TIPOS DE VARIABLES | |
|---|---|
| **TIPO** | **COMENTARIO** |
| BOOLEAN | Sólo admite 2 valores TRUE o FALSE |
| BYTE | Admite valores entre 0 y 255 |
| INTEGER | Admite valores entre -32768 y 32767 |
| LONG | Admite valores entre -2.147.483.648 y 2.147.483.647 |
| SINGLE | Admite valores decimales con precisión simple |
| DOUBLE | Admite valores decimales de doble precisión |
| CURRENCY | Válido para valores de tipo moneda |
| STRING | Cadenas de caracteres |
| DATE | Fechas, permite operar con ellas |

Figura 19

**Registros o Estructuras**

Son tipos de datos definidos por el usuario. Es un conjunto de varios datos de tipos elementales agrupados bajo una denominación común. Debe declararse en la sección **Declaraciones Generales** de un módulo. Se usa la palabra reservada Type. **Sintaxis**

Type NombreDelNuevoTipo

```
        MombreDelElemento1 As TipoDato
        MombreDelElemento3 As TipoDato
        ...
    End Type
```

| OPERADORES ARITMÉTICOS DE VISUAL BASIC | |
|---|---|
| ^ | Exponenciación |
| * | Multiplicación |
| / | División |
| Mod | Residuo entero (Ejem. A Mod B) |
| + | Suma |
| - | Resta |
| & | Concatenación de cadenas |

Figura 20

| OPERADORES DE COMPARACION | |
|---|---|
| = | Igual |
| <> | Distinto |
| > | Mayor que |
| > | Menor que |
| >= | Mayor o igual |
| <= | Menor o igual |
| Like | Compara dos cadenas<br>* Cero o más caracteres (Ejemplo: cad Like “ma”<br>? Cualquier carácter<br># Cualquier dígito (0-9)<br>**[lista]** cualquier caracter en lista<br>**[ilista]** cualquier caracter que no está en lista |
| Is | Usado para comparar dos variables de referencia a objetos. |

Figura 21

| OPERADORES LÓGICOS | |
|---|---|
| And | "Y" lógico |
| Or | "O" lógico |
| Xor | "O" Exclusivo |
| Not | Negación |

Figura 22

**Declaración de Matrices**

Para declarar matrices debemos colocar entre paréntesis el número de elementos de los que constará a continuación del nombre de la variable:

*Ejemplo:* **Dim** medidas(9) ***as*** integer

De esta forma tenemos una matriz de 10 elementos identificados del 0 al 9. Podemos obligar a que el primer elemento de una matriz tenga índice con valor 1. Esto lo haremos colocando la instrucción ***option base 1*** en la sección declarations de nuestro formulario.

También podemos indicar los límites inferior y superior de la matriz:

**Dim** medidas(5 to 14) ***as*** integer es una matriz de 10 elementos cuyos índices van del 5 al 14.

Las matrices multidimensionales se declaran de la siguiente forma:

***Dim*** medidas(1 to 10, 1 to 10) ***as*** integer

**Estructuras de Control**

Las estructuras de control permiten controlar el flujo del programa. Tenemos dos tipos de estructuras de control:

- Estructuras de Decisión
- Estructuras de Bucle

***Estructuras de Decisión***

Los elementos de Visual Basic pueden probar condiciones y, dependiendo de los resultados, realizar diferentes operaciones.

Entre las estructuras de decisión que acepta Visual Basic se incluyen las siguientes:

- ✓ If...Then
- ✓ If...Then...Else
- ✓ Select Case

*Estructura If … Then*

Se debe usar la estructura **If…Then** para ejecutar una o más instrucciones basadas en una condición. Se puede utilizar la sintaxis de una línea o un bloque de varias líneas:

**If** condición **Then** Sentencias

**If** condición **Then**

Sentencias

**End If** condicion

La *condición* normalmente es una comparación, pero puede ser cualquier expresión que dé como resultado un valor numérico. Visual Basic interpreta este valor como True o False; un valor numérico cero es False y se considera True cualquier valor numérico distinto de cero. Si *condición* es True, Visual Basic ejecuta todas las sentencias que siguen a la palabra clave **Then**. Se puede utilizar sintaxis de una línea o de varias líneas para ejecutar una sentencia basada en una condición.

*Estructura If Then Else*

Podemos utilizar un bloque **If...Then...Else** para definir varios bloques de sentencias, uno de los cuales se ejecutará:

**If** condición1 **Then**

**[**Bloque de sentencias1**]**

**[ElseIf** condición2 then**]**

**[**Bloque de sentencias2**] ...**

**[Else**

**[**Bloque de sentencias n**]]**

**End If**

*Estructura Select Case*

Esta sentencia permite realizar operaciones diferentes dependiendo del valor de una variable de Estructura General:

***Select Case*** dato

***Case*** valor1

bloque de sentencias

***Case*** valor2

bloque de sentencias

***Case*** valor3

bloque de sentencias

***Case else***

bloque de sentencias, se ejecutan si no se cumple ninguno de los anteriores

***End Select***

En esta construcción, dependiendo del valor de la variable dato se ejecutará un bloque de sentencias diferente. Los valores que podemos colocar en lugar de valor1, valor2, valor3 no sólo se limitan a valores constantes como números y cadenas de texto, sino que podemos comparar con un número.

***Estructuras de repetición o Bucle***

Las estructuras de repetición o bucle permiten ejecutar una o más líneas de código repetidamente. Las estructuras de repetición que acepta Visual Basic son:

- Do ... Loop
- For ... Next
- For Each ... Next

*Do... Loop*

Se puede utilizar el bucle **Do** para ejecutar un bloque de sentencias un número indefinido de veces. Hay algunas variantes en la sentencia **Do ... Loop**, pero cada una evalúa una condición numérica para determinar si continúa la ejecución. Como ocurre con **If ... Then**, la condición debe ser un valor o una expresión que dé como resultado False (cero) o True (distinto de cero).

Otra variante de la instrucción **Do ... Loop** es que ejecuta las sentencias primero y prueba la condición después de cada ejecución.

Hay otras dos variantes análogas a las dos anteriores, excepto en que repiten el bucle siempre y cuando la condición sea False en vez de True.

Hace el bucle cero o más veces

**Do until** condición

    Sentencias

**Loop**

Hace el bucle al menos una vez

**Do**

    Sentencia

**Loop Until** condición.

*For ... Next*

Los bucles **Do** funcionan bien cuando no se sabe cuántas veces se necesitará ejecutar las sentencias del bucle. Sin embargo, cuando se sabe que se va a ejecutar las

sentencias un número determinado de veces, es mejor elegir el bucle **For ... Next**. A diferencia del bucle **Do,** el bucle **For** utiliza una variable llamada *contador* que incrementa o reduce en cada repetición del bucle. La sintaxis es la siguiente:

**For** contador = iniciar **To** finalizar **[Step** incremento**]**

Sentencias

**Next [**contador**]**

*For Each ... Next*

El bucle **for Each … Next** es similar al bucle **For ... Next,** pero repite un grupo de sentencia por cada elemento de una colección de objetos o de una matriz en vez de repetir las sentencias un número especificado de veces. Esto resulta especialmente útil si no se sabe cuántos elementos hay en la colección. He aquí la sintaxis :

**For Each** elemento **in** grupo

Sentencias

**Next** elemento

**Características Generales de los Controles en Visual Basic**

- ✓ ***Propiedades:*** Todos los controles disponen de una serie de propiedades las cuales podemos cambiar al incluirlos en nuestras aplicaciones. Ejemplos de propiedades son el color, el tipo de letra, el nombre, el texto, etc.

  Las propiedades más comunes que son aplicadas a los formularios y controles son las siguientes:

*Name:* Nombre, necesario para llamar al formulario o control desde el código.

*Caption:* Texto que aparece en el título.

*Backcolor:* Color de fondo.

*Forecolor:* Color del texto del formulario.

- ✓ ***Métodos:*** Son procedimientos asociados a los controles, es decir, rutinas ya establecidas que podemos invocar desde nuestras aplicaciones para que se realice alguna operación sobre el control. Por ejemplo el control ListView ( la lista de archivos que aparece en el explorador de windows) dispone del método order que te ordena los datos aparecidos en la lista.

*Método Show*

Para llamar a un formulario desde el código utilizaremos el método Show. Si el formulario 2 tiene en la propiedad *Name form2***,** para llamarlo desde otro formulario se debe hacer lo siguiente: Form2.Show

Si no ponemos ningún argumento se asume que el formulario aparece en modo no modal, es decir, se permitirá que se active cualquier otro formulario sin cerrar el formulario 2. La otra modalidad que existe es modal, lo que significa que no se permite el enfoque hacia ningún otro formulario hasta que no se cierre el actual. Este último modo puede servir para cuadros de diálogo que soliciten ciertos parámetros para que la aplicación siga funcionando: una contraseña.

- ✓ ***Eventos****:* Son acciones que pueden ser motivadas por el propio usuario o por el mismo sistema operativo. Ejemplos pueden ser el movimiento del ratón o hacer clic en su botón.

Para mostrar la ventana donde aparecen los controles que Visual Basic carga por defecto nada más se debe arrancar la aplicación y marcar en ***View*** del menú principal (versión inglesa) y activar la opción ***Toolbox.***

Realmente existen muchos más controles, aunque estos son los más utilizados y por eso aparecen por defecto. Para tener acceso a los demás controles tanto de Visual Basic como los controles que incorporan otras aplicaciones marcaremos en ***Tools/Custom Controls*** del menú principal.

Para que esta ventana aparezca siempre en primer plano aunque no sea la ventana activa marcaremos con el botón derecho del ratón en cualquier lugar de la ventana y activaremos la opción ***Always On Top*** del menú contextual.

Para acceder a los controles de otro formulario se sigue la siguiente sintaxis:

Formulario!Control.Propiedad = valor

Se debe colocar un signo de admiración "!" entre el formulario y el control, y un punto entre el control y la propiedad

Ejemplo:

Form2!Label1.Caption = "Número de Clientes"

Al acceder a las propiedades de otro formulario automáticamente se carga éste en memoria, si no lo estaba ya antes. Una vez que se han modificado sus propiedades se visualizan con el método Show.

No se puede acceder a las variables declaradas en otro formulario, de modo que si se desea trabajar con variables generales, las cuales son accesibles desde cualquier formulario de la aplicación, se tienen que declarar como *Públicas* desde un *módulo de código.*

Para insertar un *módulo* en el proyecto de Visual Basic se debe marcar en ***Insert/ Module***. Aparece una ventana en la que únicamente se puede colocar las variables y procedimientos o funciones que se quiere sean públicas para toda la aplicación.

**Controles de Bases de Datos**

Sin una sola línea de código se puede visualizar cualquier tabla de una base de datos con todos sus registros. Únicamente hacen falta 2 controles:

***Data:*** este control viene en la caja de herramientas (toolbox) nada más se debe cargar Visual Basic. Es un control muy potente, el más importante para bases de datos. Permite abrir una base de datos y manipular su contenido: situarnos en un registro concreto de una tabla, eliminar, añadir o modificar registros, etc.

***DBgrid:*** Si el control no se encuentra inicialmente disponible en la toolbox, se tiene que buscar en la opción *Custom Controls* del menú Tools (si se está utilizando la versión inglesa), aparecerá como 1ª opción de las muchas que existen *Apex Data Bound Control*, activar y aceptar. Este control permite visualizar un conjunto de registros, cabeceras incluidas, que hayan sido indicado en un control *Data.*

Con el *DataBaseName* se da el nombre donde está situado el archivo que contiene la base de datos.

*RecordSource:* Aparecerán las tablas de las que consta la base de datos para que elijamos con cual queremos trabajar. También se puede indicar mediante una sentencia SELECT campos de distintas tablas que cumplan ciertas condiciones.

*DataSource:* Esta propiedad permite proporcionar la fuente o base de datos.

Para administrar la información de la base de datos usamos algunas sentencia especiales como la siguiente:

*Sentencia Select:* Permite seleccionar registros de una o varias tablas que cumplan con ciertas condiciones para, posteriormente, realizar las operaciones que deseemos con los registros seleccionados, mostrar, modificar, eliminar, añadir, etc.

Una sentencia SELECT se compone de varias palabras clave: FROM, WHERE, GROUP BY, HAVING y ORDER BY.

Las únicas obligatorias son SELECT y FROM, las demás son opcionales aunque su uso añade una mayor potencia a la sentencia.

*Opciones de SELECT*

Lo primero que se debe indicar son los campos que aparecerán en la selección:

SELECT campo1, campo2, ....., campoN

Si se quieren obtener todos los campos de la tabla o tablas seleccionadas se indica con un asterisco.

SELECT *

Si el campo es compuesto se pone entre corchetes: SELECT Nombre, [Año Nacimiento]

También se puede asignar un alias a cualquier campo, ese alias será el nombre con el que aparecerá el campo en la selección. Por ejemplo: SELECT [Año Nacimiento] AS FechaNac . En este caso es más cómodo trabajar con FechaNac.

Si se realiza la selección entre varias tablas que contienen campos que se llaman igual, debemos anteponer la tabla de la que proceden si se quiere que alguno de esos campos aparezcan en la selección:

SELECT Empresas.codigo, Empleados.codigo FROM Empresas, Empleados

Al estar repetido el campo código en las dos tablas se indica la tabla de la que proceden. Aquí se pone la cláusula FROM que debe aparecer en todas las SELECT, su sintaxis es muy simple, ya que tan solo hay que indicar los nombres de la tabla o tablas sobre las que se hace la consulta.

ORDER BY: Permite indicar el campo o campos por los que se ordenarán los registros, y si lo hacen de forma ascendente o descendente:
SELECT Apellidos, Nombre FROM Empleados ORDER BY Apellidos, Nombre DESC

Se obtiene una consulta de los empleados ordenada por Apellidos y Nombre de forma descendente, si dos empleados tienen los mismos apellidos (deben de ser familiares), se ordenarán también por el Nombre.

También se puede ordenar los registros por campos que no aparezcan en la consulta, y si se quiere ordenar de forma ascendente no hace falta que se ponga ninguna palabra ya que es como se ordenan de forma predeterminada, de todas formas la cláusula es ASC.

DISTINCT: Sirve para obtener los campos que se desea, sin mostrar los valores repetidos.

SELECT DISTINCT departamentos FROM empleados

DISTINCTROW: Mediante esta cláusula se obtienen los registros de una tabla o selección de tablas que no se repitan. En este caso no se debe repetir ningún campo de todos los que componen el registro con lo cual esta cláusula es menos restrictiva que DISTINCT en la que sólo aparecen los registros en los campos que se indiquen no se repitan.

TOP: Devuelve el número de registros indicados situados en la parte superior o inferior según el orden de los registros con ORDER BY.

SELECT TOP 10 Nombre FROM Empleados ORDER BY Codigo

Con PERCENT se le indica que devuelva un porcentaje en lugar de un número exacto de registros.

SELECT TOP 5 PERCENT NombreDepto FROM Deptos ORDER BY Codepto DESC

A pesar que existen diversos DBMS y Software disponibles para la construcción de la aplicación, seleccionamos ACCESS y VISUAL BASIC 6.0 por las siguientes razones:

- Disponibilidad de licencias del software por parte de la Institución.
- Compatibilidad con el plan de desarrollo de aplicaciones por parte del Departamento de Informática de la Institución en proyectos que actualmente de están desarrollando.
- Facilidad de migración de datos desde las tablas que actualmente se utilizan, a saber, FOX para DOS.

- Ambiente de desarrollo orientado a objetos compatible con otros sistemas que utiliza el hospital y que permite una interfaz adecuada.

**Análisis Costo-Beneficio**

A continuación se determina la factibilidad económica del sistema utilizando el método del valor presente, es importante mencionar que por ser el hospital una institución de carácter social sin fines de lucro, la evaluación económica se realiza a través de una estimación de ahorro en tiempos de algunos eventos que se consideran relevantes y que ha sido posible cuantificar en esta etapa.

Estimación de vida útil del sistema: 5 Años.

Promedio de Inflación anual: 1.4 % (Según el BCR para los próximos 5 años)

*Costo inicial del proyecto*

- Inversión Inicial:

| | |
|---|---|
| 5 estaciones de trabajo con valor de | $ 2500.00 |
| 1 Hub 16 puertos | $ 80.00 |
| 2 Impresores Laser | $ 700.00 |
| 1 Impreso matricial | $ 422.86 |
| Total: | $ 3702.86 |

*Costos de funcionamiento*

- Mantenimiento preventivo anual: $ 1,234.80
- Depreciación anual del equipo: $ 656.00

- Costo energía eléctrica anual : 503 kw x 1.18 = $ 593.52

Total anual : $ 2484.34

Total costos de funcionamiento con inflación:

Año 2003 : 2484.34 x 0.014 = 2,519.12

Año 2004 : 2519.12 x 0.014 = 2,554.38

Año 2005 : 2554.38 x 0.014 = 2,590.14

Año 2006 : 2590.14 x 0.014 = 2,626.40

Total: $ 10,290.04

Ahorros anuales con la implementación del Sistema:

A partir de los datos recopilados en las encuestas y la observación directa en el área, podemos ver el tiempo que se utiliza en realizar cada proceso. Con la utilización del nuevo sistema podemos comparar tiempos con sus costos y determinar el ahorro que esto significa para la Institución.

*Requerir personal desde el banco de Currículos:*

Tiempo utilizado con el antiguo proceso: 480 Min.

Tiempo utilizado con el nuevo proceso: 120 Min.

Sueldo del empleado: $350.00

Pago por proceso manual: 11.66 Total 5 años $ 8,278.60

Pago en el nuevo proceso: 2.91 Total 5 años $ 2,066.10

Total de Ahorro: $ 8.75

Número de procesos en el año: 142 en cinco años: 710

Ahorro anual en el proceso: $ 1242.50 Ahorro en cinco años: $ 6,212.50

*Emisión de Constancias de tiempo y salario:*

Tiempo utilizado con el antiguo proceso: 30 Min.

Tiempo utilizado con el nuevo proceso: 5 Min.

Sueldo del empleado: $350.00

Pago por proceso manual: 0.73 Total 5 años $5,256.00

Pago en el nuevo proceso: 0.12 Total 5 años $ 864.00

Total de Ahorro: 0.61

Cantidad constancias anual: 1,440 en 5 años: 7,200

Ahorro anual en el proceso: $ 878.40 Ahorro en cinco años: $ 4,392.00

*Creación de Registro de Contratación:*

Tiempo utilizado con el antiguo proceso: 180 Min.

Tiempo utilizado con el nuevo proceso: 45 Min.

Sueldo del empleado: $350.00

Pago por proceso manual: 4.37 Total 5 años $8,608.90

Pago en el nuevo proceso: 1.09 Total 5 años $2,147.30

Total de Ahorro: $ 3.28

Total ingreso de personal : 394 en 5 años: 1,970

Ahorro anual en el proceso: $ 1,292.32 Ahorro en cinco años: $ 6,461.60

Gasto con Sistema Manual en 5 años: $ 22,143.50

Gastos con Sistema Mecanizado en 5 años: $ 15,367.14 + $3702.86 (Inv.Inicial)

Nota: Esta es una muestra de ahorro tomando solamente tres procesos.

**Requerimientos mínimos para Instalación y utilización de Visual Basic**

- ✓ PC con procesador Pentium a 166 MHz o superior (Pentium III recomendado).
- ✓ Sistema Operativo Microsoft Windows 95 o Microsoft Windows NT 4.0 con Service Pack 3 (Service Pack 3 incluido) o posterior.
- ✓ 64 MB en RAM para Windows 95 y Windows NT .
- ✓ Unidad CD ROM.
- ✓ Monitor Super VGA.
- ✓ Espacio en disco duro requerido:
  - Instalación típica: 116 MB.
  - Instalación máxima: 135 MB.

Espacio adicional requerido en disco duro para los siguientes productos:

MSDN: 57 MB típica, 493 MB máxima.

Windows NT 4.0 Option Pack: 20 MB para Windows 95 o posterior; 200 MB para Windows NT 4.0. o mayor.

## 1.4 Marco Legal

El marco legal fundamentalmente se basa en disposiciones legales que de manera directa o indirecta tienen influencia en el diseño de la aplicación.

En el caso particular del proyecto, para el diseño de las estructuras y los procesos se toma en cuenta una LEY DE SALARIOS que es emitida por el gobierno anualmente con sus respectivas modificaciones. Cada año, las plazas asignadas a los empleados sufren cambios radicales. Estos cambios son plasmados en un documento llamado REORGANIZACION ANUAL donde se especifican nuevas partidas y subnúmeros de partidas a los empleados. Basados en ésta codificación crecen todos los registros relacionados al empleado junto a su plaza.

## 1.5 Marco Conceptual

A continuación presentamos algunos de los términos utilizados en el primer capítulo del proyecto para presentar como trabajo de grado y que consideramos deben tener un significado claro para el lector.

*Seguimiento de empleados:* Control administrativo de eventos laborales documentados, entre estos están los permisos para no asistir a trabajar, renuncias, incapacidades, etc.

*Multiusuario:* Software que permite la interacción simultanea con varios usuarios en diferentes estaciones de trabajo Cliente/Servidor.

*Estandarización:* Criterio común establecido por acuerdo y que sirve para establecer un procedimiento.

*4GL:* Lenguaje de cuarta generación basado en la interacción con bases de datos.

*Sistematización:* Es la técnica de ordenar, crear o modificar los flujos, procesos y almacenamiento de datos manuales y mecanizados, utilizando los recursos adecuados con el fin de obtener mejores resultados en la búsqueda de un objetivo dado.

*Interfaz:* Es el medio que se utiliza en las aplicaciones para la interacción con el usuario.

*Software:* Programas y rutina que indican a la computadora que hacer y cuando hacerlo.

*Modulo:* Unidad en un esquema de empaque. Subsistema con sus propias características y que forma parte de otro sistema macro guardando una estrecha relación con él.

*Implementación:* Llamaremos implementación en muestro proyecto a la prueba funcional del software listo para utilizar y con previas pruebas exhaustivas de verificación en su consistencia.

*Estructuras Jerárquicas:* Estructuras con sus propias características pero dependiendo de otras en niveles jerárquicos.

*Retroalimentación:* Proceso que permite que la salida afecte la entrada.

*UML:* Lenguaje de Modelación Unificada.

*DBMS:* Sistema Administrador de Bases de Datos.

*SGBD:* Sistema Gestor de Bases de Datos Relacionales

*Hito:* Delimitador en un proceso o fase específico.

*Instancias:* Relaciones que existen entre clases trasladando la herencia entre ellas.

*Casos de Uso:* Diagramas que muestran la forma como se realiza una actividad y donde participan entidades o actores.

*Entidad:* Se hace referencia como entidad a una persona, almacenamiento, sistema o institución.

*Dominio:* Llamaremos dominio al conjunto de valores que puede tener un atributo, también se utiliza como la delimitación de actividades que serán puestas en análisis para definir las clases.

*DFD:* Diagrama de flujo de datos. En nuestro caso se utilizarán exclusivamente para mostrar las relaciones entre subsistemas solamente.