CAPÍTULO DOS
MARCO TEÓRICO

2.1 CARACTERÍSTICAS, VENTAJAS Y DESVENTAJAS DE LAS BASES DE DATOS RELACIONALES.

2.1.1 Definición de base de datos.

La base de datos puede definirse como una colección de datos interrelacionados almacenados en conjunto, sin redundancias perjudiciales o innecesarias; su finalidad es de servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados .Dícese que un sistema comprende una colección de base de datos cuando éstos son totalmente independientes desde el punto de vista estructural.

2.1.2 Base de datos relacional.

Una base de datos relacional es aquella que cumple con el modelo relacional, el cual es el más utilizado en la actualidad para implementar bases de datos ya planificadas. Permite establecer interconexiones (relaciones) entre los datos (que están guardados en tablas) y a través de dichas conexiones relacionar los datos de las tablas, de ahí proviene su nombre: "Modelo Relacional".

Los sistemas relacionales son importantes porque ofrecen ventajas tales como: simplicidad, generalidad y las consultas de información se especifican de forma sencilla. Las tablas son un medio para organizar la información de una forma más compacta y al mismo tiempo acceder a ella desde una interfaz.

2.1.3 Características de las bases de datos relacionales

- Las bases de datos relacionales están constituidas por una o más tablas que contienen la información ordenada, organizada y que además cumplen con las siguientes características básicas:
  - Una tabla sólo contiene un ID único e irrepetible.
  - El nombre de los campos de una tabla es distinto.
  - Cada registro de la tabla es único.
  - El orden de los registros no está determinado.
  - Para cada campo existe un conjunto de valores posible.

- La planificación de la estructura de la base de datos, en particular de las tablas es vital para la administración. Cada tabla tiene una llave primaria, un identificador único, compuesto por una o más columnas. La mayoría de las llaves primarias están formadas por una única columna (atributo), es decir, la llave primaria es un conjunto de uno o más atributos que sirve para identificar de forma univoca a una tupla.

Para establecer la relación entre dos tablas es necesario incluir, en forma de columna, en una de ellas la llave primaria de la otra. A esta columna se le llama llave secundaria. La llave secundaria nos permite expresar relaciones entre objetos.

- Integridad Referencial es otra característica importante. Se entiende por integridad referencial a las reglas que se establecen para mantener las relaciones entre las tablas cuando se agregan, cambian o eliminan registros.

Al exigir la integridad referencial, se impide a los usuarios que agreguen registros a la tabla relacionada para la cual no hay llave principal, cambiar los valores de la tabla principal que darían lugar a registros huérfanos en la tabla relacionada, así como eliminar registros de una tabla principal cuando hay registros relacionados coincidentes.

Los términos formales del modelo relacional a menudo son sustituidos por otros de uso común, debido a que estos términos son demasiado abstractos para ser usados en la práctica. La relación se muestra en la tabla 2.1.1.

Término relacional formal	**Equivalente informal**
Relación	Tabla
Tupla	Fila o Registro
Cardinalidad	Número de filas o registros
Atributo	Columna o campo
Grado	Número de columnas o campos
Llave Primaria	Identificador único
Dominio	Conjunto de valores permitidos para el atributo.

Tabla 2.1.3.1 - Términos formales e informales del modelo relacional.

2.1.4 Cardinalidad de las relaciones

El tipo de cardinalidad se representa mediante una etiqueta en el exterior de la relación, respectivamente: "1:1", "1:N" y "N:M", aunque la notación depende del lenguaje utilizado, la que más se usa actualmente es el unificado. Otra forma de expresar la cardinalidad es situando un símbolo cerca de la línea que conecta una entidad con una relación:

- **"0"** si cada instancia de la entidad no está obligada a participar en la relación.

- **"1"** si toda instancia de la entidad está obligada a participar en la relación y, además, solamente participa una vez.
- **"N", "M", o "*"** si cada instancia de la entidad no está obligada a participar en la relación y puede hacerlo cualquier número de veces.

2.1.5 Algebra relacional

El álgebra relacional es un conjunto de operaciones que describen paso a paso como computar una respuesta sobre las relaciones, tal y como éstas son definidas en el modelo relacional, denominada de tipo procedural.

Describe el aspecto de la manipulación de datos. Estas operaciones se usan como una representación intermedia de una consulta a una base de datos y debido a sus propiedades algebraicas, sirven para obtener una versión optimizada y eficiente de dicha consulta.

2.1.5.1 Operaciones básicas

Cada operador del álgebra acepta una o dos relaciones y retorna una relación como resultado. σ y Π son operadores unarios, el resto de los operadores son binarios. Las operaciones básicas del álgebra relacional son:

- **Selección (σ)**

Permite seleccionar un subconjunto de tuplas de una relación (R), todas aquellas que cumplan la(s) condición(es) P, esto es:

$\sigma_P(R)$

- **Proyección (Π)**

Permite extraer columnas (atributos) de una relación, dando como resultado un *subconjunto vertical* de atributos de la relación, esto es:

PA1,A2,……An(R)
donde, *A1,A2,……An* son atributos de la relación R .

- **Producto cartesiano (x)**

El producto cartesiano de *dos relaciones* se escribe como:

$R \times S$

y entrega una relación, cuyo *esquema* corresponde a una combinación de todas las tuplas de R con cada una de las tuplas de S,y sus atributos corresponden a los de R seguidos por los de S.

- **Unión (∪)**

La operación

$R \cup S$

retorna el conjunto de tuplas que están en R, o en S, o en ambas. R y S deben ser *uniones compatibles*.

- **Diferencia (-)**

La diferencia de dos relaciones, R y S denotada por:

$R - S$

entrega todas aquellas tuplas que están en R, pero no en S. R y S deben ser *uniones compatibles*.

Estas operaciones son fundamentales en el sentido en que (1) todas las demás operaciones pueden ser expresadas como una combinación de éstas y (2) ninguna de estas operaciones pueden ser omitidas sin que con ello se pierda información.

2.1.5.2 Operaciones no básicas

- **Intersección (∩)**

La intersección de dos relaciones se puede especificar en función de otros operadores básicos:

$$R \cap S = R - (R - S)$$

La intersección, como en Teoría de conjuntos, corresponde al conjunto de todas las tuplas que están en R y en S, siendo R y S *uniones compatibles*.

- **Unión natural (⋈) (Natural Join)**

La operación unión natural en el álgebra relacional es la que permite reconstruir las tablas originales previas al proceso de normalización. Consiste en combinar las proyección, selección y producto cartesiano en una sola operación, donde la condición θ es la igualdad Clave Primaria = Clave Externa (o Foranea) y la proyección elimina la columna duplicada (clave externa).

Expresada en las operaciones básicas, queda

$$R \bowtie S = \Pi_{A1,A2...An}(\sigma_\theta(R \times S))$$

Una reunión theta (θ-Join) de dos relaciones es equivalente a:

$$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

donde la condición θ es libre.

Si la condición θ es una igualdad se denomina EquiJoin.

- **División (/)**

Supongamos que tenemos dos relaciones A(*x, y*) y B(*y*) donde el dominio de *y* en A y B, es el mismo.

El operador división A / B retorna todos los distintos valores de *x* tales que para todo valor *y* en B existe una tupla $\langle x, y \rangle$en A.

- **Agrupación (Ǥ)**

Permite agrupar conjuntos de valores en función de un campo determinado y hacer operaciones con otros campos.

2.1.6 Cálculo Relacional

El Cálculo relacional es un lenguaje de consulta que describe la respuesta deseada sobre una Base de Datos sin especificar cómo obtenerla, a diferencia del álgebra relacional que es de tipo procedural, el cálculo relacional es de tipo declarativo, pero siempre ambos métodos logran los mismos resultados.

Podemos distinguir, al menos, dos clases de cálculo relacional:

- **Cálculo relacional basado en tuplas. (TRC).**

Una consulta en TRC es de la forma:

$\{T \mid \varphi(T)\}$

donde T es una variable tipo tupla y φ(T) es una fórmula que describe a T. El resultado de esta consulta es todas las tuplas t para las cuales la fórmula es verdadera.

- **Cálculo relacional basado en dominios (DRC)**

Está constituido con los mismos operadores que el cálculo relacional de tuplas pero no hay tuplas sino variables dominio. Las expresiones del cálculo relacional de dominios

son de la forma { (x, y, z, ...) / P(x, y, z, ...) }. Donde x, y, z representan las variables de dominio, P representa una fórmula compuesta de átomos (igual que en el CRT). Los átomos del cálculo relacional de dominios tienen una de las siguientes formas:

- (x, y, z) $\in$ r, donde r es una relación con n atributos y x , y, z son variables de dominio o constantes.
- x θ y, donde x e y son variables de dominio y θ es un operador de comparación aritmética (>, <, =, $\neq$). Es necesario que los atributos x e y, tengan dominios cuyos miembros puedan compararse medianteθ.
- x θ c, donde x es una variable de dominio, θ es un operador de comparación y c es una constante en el dominio del atributo x.

2.1.7 Normalización.

La normalización es un proceso que pretende conseguir tablas con una estructura óptima y eficaz. El proceso de normalización está basado en lograr la independencia de los datos respecto a las aplicaciones que los usan.

Antes de empezar el proceso, se han de conocer las tablas que intervendrán y las relaciones que las unen. Si no se conocen a partir del análisis previo, se buscan todos los nombres (sustantivos) que han sido empleados en la definición del problema. Algunos de esos nombres serán las entidades, otros dependerán de ellas y serán los atributos. Otros no formarán parte ni de las entidades ni de los atributos, son parte del lenguaje necesario para describir el problema a solucionar mediante la creación de una base de datos.

El proceso de normalización se basa en la descomposición sin pérdida de las tablas que están en una forma normal inferior, obteniéndose una forma normal superior. El

proceso de descomposición sin pérdida, significa que se ha de dividir o descomponer la tabla en otras con menor cantidad de atributos sin que haya pérdida de información.

La teoría de la normalización está basada en el concepto de formas normales. Se dice que una relación está en una forma normal particular si satisface cierto conjunto específico de restricciones.

2.1.7.1 Formas normales y dependencias funcionales.

Las formas normales son aplicadas a las tablas de una base de datos. Decir que una base de datos está en la forma normal **N** es decir que todas sus tablas están en la forma normal **N**. En general, las primeras tres formas normales son suficientes para cubrir las necesidades de la mayoría de las bases de datos.

- Primera forma normal ó 1FN:

La Primera forma normal, ó 1FN, es la más elemental de todas. Una tabla está en 1FN si el valor que contiene un atributo de un registro, un campo, es único y elemental. En cada uno de los atributos sólo se puede incluir un dato, pero no se pueden incluir una lista de datos.

- Segunda forma normal ó 2FN:

Se dice que un atributo o conjunto de atributos tiene dependencia funcional de otro u otros si a cada uno de los primeros le corresponde sólo uno de los segundos.

Una tabla está en segunda forma normal ó 2FN cuando está en 1FN y todo atributo que no pertenece a la clave primaria tiene una dependencia funcional de la clave completa y no de parte de ella. Luego, si la clave principal está formada por un solo atributo y ya está en 1FN, ya estará en 2FN.

- Tercera forma normal ó 3FN:

Se dice que hay dependencia funcional transitiva entre dos atributos cuando un atributo que no pertenece a la clave primaria permite conocer el valor de otro atributo.

Una tabla está en tercera forma normal ó 3FN si está en 2FN y no existen atributos que no pertenezcan a la clave primaria que puedan ser conocidos mediante otro atributo que no forma parte de la clave primaria, es decir, no hay dependencias funcionales transitivas.

- Forma normal de Boyce-Codd o FNBC:

Una tabla está en Forma Normal de Boyce-Codd o FNBC si solo existen dependencias funcionales elementales que dependan de la clave primaria o de cualquier clave alternativa. Si la clave primaria está formada por un solo atributo y está en 3FN, ya está en FNBC.

- Cuarta forma normal ó 4FN:

Existe dependencia funcional multivalorada o de múltiples valores si, dados tres atributos de una tabla, si para cada valor del primer atributo existen múltiples valores en el segundo atributo y no hay ninguna relación entre el tercer atributo y el primero, a no ser a través del segundo atributo.

Una tabla está en cuarta forma normal ó 4FN si está en FNBC y las únicas dependencias funcionales multivaloradas que existen son las dependencias funcionales de la clave con los atributos que no forman parte de la misma. Estas dependencias multievaluadas de la clave con los atributos que no forman parte de la misma son dependencias triviales, por lo que algunos autores dicen que no existen dependencias multievaluadas en 4FN.

- Quinta forma normal ó 5FN:

Se dice que hay dependencia de JOIN, de unión o de producto si una tabla tiene dependencia de *unión con varias de sus *proyecciones y se puede obtener la tabla por medio de la unión de dichas proyecciones.

Una tabla esta en quinta forma normal (5FN) o Forma Normal de Proyección-Unión si está en 4FN y las únicas dependencias que existen son las dependencias de unión de una tabla con sus proyecciones relacionándose entre las distintas proyecciones mediante la clave primaria o cualquier clave alternativa. La 5FN se emplea cuando en una misma tabla tenemos mucha información redundante, con pocos atributos o cuando una tabla posee una gran cantidad de atributos y se hace por ello inmanejable.

2.1.8 Ventajas de las bases de datos relacionales.

Las bases de datos relacionales presentan numerosas ventajas:

- Evitan la redundancia de datos. En las bases de datos relacionales, los datos están integrados, por lo que no se almacenan varias copias del mismo dato.

- Consistencia de datos. Al controlar la redundancia, se reduce en gran medida el riesgo de que haya inconsistencia.
- Concurrencia de datos. La base de datos pertenece a la empresa y puede ser utilizada por todos los usuarios autorizados en forma simultánea.
- Mantener estándares. Facilita el intercambio de datos y la administración de los mismos.
- Integridad de datos. A través de reglas que no pueden ser violadas. Las restricciones se aplican tanto a datos, como a sus relaciones.
- El uso de una base de datos relacional bien diseñada puede reducir mucho la cantidad de datos que debe ingresar cada vez que se agrega un registro. Para un número grande de registros, una base de datos relacional puede buscar más rápido entre los mismos.
- Puede crear formularios e informes que muestren solo los datos que se quiere ver.
- Garantiza la integridad referencial, al eliminar un registro todos los registros relacionados dependientes también se eliminan.
- Favorece la normalización por ser más comprensible y aplicable.

2.1.9 Desventajas de las bases de datos relacionales.

- Se tienen que hacer varios registros, ya que si se tiene una base de datos centralizada, se puede perder la información.
- No se manipulan de forma amigable los bloques de texto como los de tipo dato.
- Presentan deficiencias con datos gráficos, multimedia, CAD (Diseño Asistido por Computadora) y sistemas de información geográfica.

- Las bases de datos orientadas a objetos (BDOO) se propusieron con el objetivo de satisfacer las necesidades de las aplicaciones anteriores y así, complementar pero no sustituir a las bases de datos relacionales.
- Incompatibilidad. Si las especificaciones se ponen por escrito, no hay problema; pero en la práctica cotidiana, las incompatibilidades mayores o menores entre computadoras, sistemas operativos, lenguajes, protocolos, interfaces y programas de aplicación superan las expectativas. Cuanto más elevado es el número de proveedores, las incompatibilidades son mayores.

2.2 CARACTERÍSTICAS, VENTAJAS Y DESVENTAJAS DEL DESARROLLO DEL SISTEMA EN EL MÉTODO DE CICLO DE VIDA CLÁSICO.

El método de Ciclo de Vida para el Desarrollo de Sistemas es el conjunto de actividades que los analistas, diseñadores y usuarios realizan para desarrollar e implantar un sistema de información.

En este método se modela el ciclo convencional de la Ingeniería del Software, aplicando un enfoque sistemático y secuencial de desarrollo. Consta de seis fases que son las siguientes: Análisis del sistema, Diseño, Instrumentación, Pruebas, Implementación y Mantenimiento. Véase figura 2.2.1.

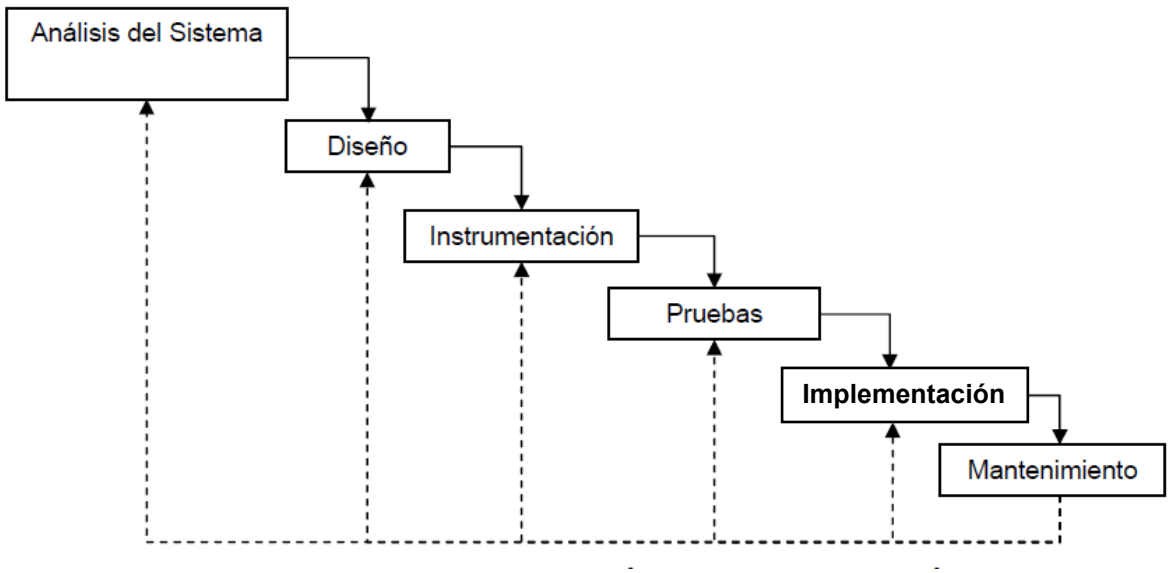

Figura 2.2.1 Etapas del Método de ciclo de vida clásico

Cada fase se efectúa mediante la aplicación de métodos explícitos, herramientas y técnicas. La introducción de una revisión y vuelta atrás, se realiza con el fin de corregir

las deficiencias detectadas durante las distintas etapas, o para completar o aumentar las funcionalidades del sistema en desarrollo. De esta manera, durante cualquiera de las fases se puede retroceder momentáneamente a una fase previa para solucionar los problemas que se pudieran haber encontrado.

2.2.1 Análisis del Sistema

El Análisis del Sistema se divide en dos subfases: Planeación y Definición de Requerimientos. Véase figura 2.2.1.1

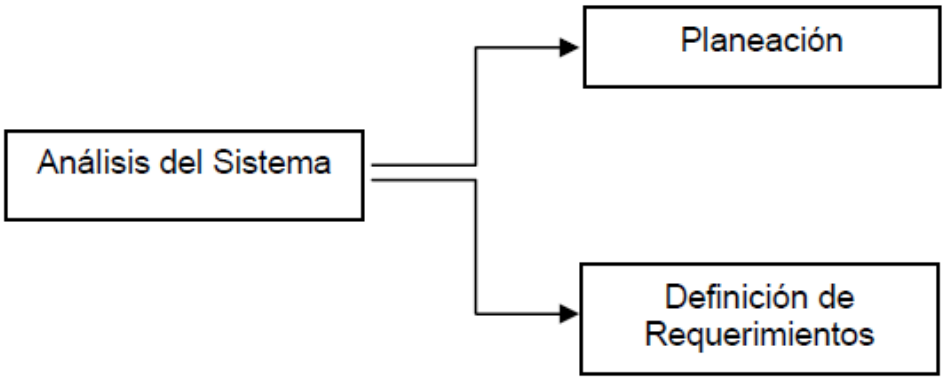

Figura 2.2.1.1 División de la Fase de Análisis del Sistema

En la subfase de Planeación se llevan a cabo las siguientes actividades:

- Comprensión del problema del cliente

Antes de considerar cualquier investigación de sistemas, la solicitud de proyecto debe examinarse para determinar con precisión lo que el solicitante desea antes de seguir adelante, la solicitud de proyecto debe estar claramente planteada.

- Estudio de Factibilidad

Un resultado importante de la investigación preliminar es la determinación de que el sistema solicitado sea factible. En la investigación preliminar existen tres aspectos relacionados con el estudio de factibilidad:

- Factibilidad Técnica:

El trabajo para el proyecto, ¿Puede realizarse con el equipo actual, la tecnología existente de software y el personal disponible? Si se necesita nueva tecnología, ¿Cuál es la posibilidad de desarrollarla?

- Factibilidad Económica:

Al crear el sistema, ¿los beneficios que se obtienen serán suficientes para aceptar los costos?, ¿los costos asociados con la decisión de *no* crear el sistema son tan grandes que se debe aceptar el proyecto?

- Factibilidad Operacional:

Si se desarrolla e implanta, ¿será utilizado el sistema?, ¿existirá cierta resistencia al cambio por parte de los usuarios que dé como resultado una disminución de los posibles beneficios de la aplicación?

- Desarrollo de la estrategia de Solución recomendada
- Determinación de los criterios de aplicación
- Planeación del proceso de desarrollo

2.2.2 Determinación de los requerimientos del sistema

El aspecto fundamental del análisis de sistemas es comprender todas las facetas importantes de la parte de la empresa que se encuentra bajo estudio. Los analistas, al

trabajar con los empleados y administradores, deben estudiar los procesos de una empresa para dar respuesta a las siguientes preguntas clave:

- ¿Qué es lo que hace?
- ¿Cómo se hace?
- ¿Con que frecuencia se presenta?
- ¿Qué tan grande es el volumen de transacciones o decisiones?
- ¿Cuál es el grado de eficiencia con el que se efectúan las tareas?
- ¿Existe algún problema? ¿Qué tan serio es? ¿Cuál es la causa que lo origina?

La subfase de Planeación se divide en: Definición del Sistema y Plan de Proyecto. Véase figura 2.2.2.1

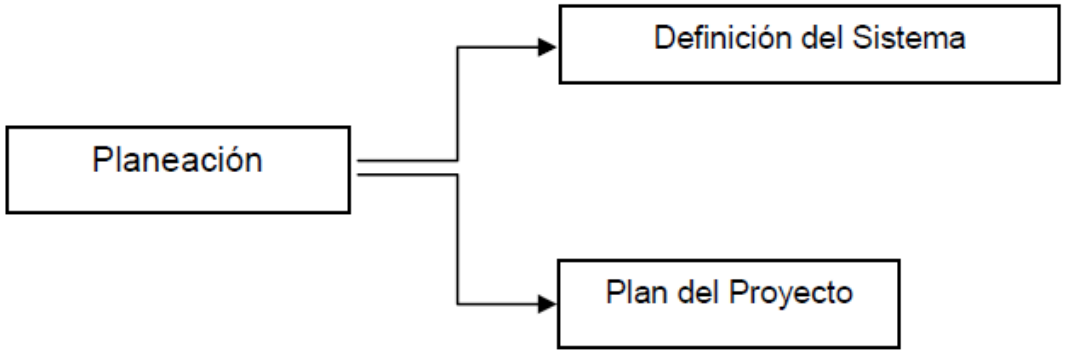

Figura 2.2.2.1 Productos de la Planeación

La Definición del Sistema puede contener cuadros, figuras, gráficas, y ecuaciones de distintos estilos. La notación exacta empleada en la Definición del Sistema depende mucho del área del problema que se trate. El documento de Definición del Sistema presenta los siguientes elementos:

- Definición del problema.
- Justificación del sistema.
- Metas del sistema y del proyecto.
- Restricciones del sistema y del proyecto.
- Funciones que se proporcionarán (equipo/programación/personal).
- Características del usuario.
- Ambientes de desarrollo/operación/mantenimiento.
- Estrategia de solución.
- Prioridades para las características del sistema.
- Criterios de aceptación del sistema.
- Fuentes de información.
- Glosario de términos.

El Plan del Proyecto contiene el modelo del ciclo de vida que se utilizará, la estructura organizacional del proyecto, la programación preliminar del desarrollo, estimados preliminares de costos y recursos, así como de personal, herramientas y técnicas que se emplearán, y estándares que se seguirán. Los elementos que se deben incluir en el Plan del proyecto son los siguientes:

- Modelo del ciclo de vida (terminología/logros/productos finales).
- Estructura organizacional.
- Estructura de administración/de equipos/distribución de trabajo/definición de puestos
- Requisitos preliminares de personal y recursos.
- Programación preliminar del desarrollo (Redes PERT/Gráficas de Gant).
- Estimado preliminar de costos.
- Mecanismos de supervisión y control del proyecto.
- Herramientas y técnicas que se emplearán.

- Lenguajes de programación.
- Requisitos de prueba.
- Plan de instalación.
- Consideraciones de mantenimiento.
- Método y tiempo de la entrega final.
- Método y tiempo de pago.
- Fuentes de información.

Durante la fase de planeación los estimados de costos y la programación del trabajo serán preliminares, debido a que usualmente no es posible realizar estimaciones precisas sin haber realizado algo del diseño. Las prácticas actuales de contratación requieren que el costo final y la programación se proporcionen durante la fase de planeación. Esta situación, aunada a la naturaleza competitiva del medio, es una de las principales razones de los excesos en costos y las entregas retrasadas de los productos de programación.

Reconociendo esta realidad, muchas organizaciones utilizan una serie sucesiva de estimaciones de costos y programación. Los estimados preliminares se preparan durante la fase de planeación, su redefinición se presenta en la revisión preliminar del diseño; el costo y la programación finales se establecen en la revisión final del diseño. Distintas estimaciones, que representan una clase de capacidades, pueden mostrarse en cada una de las revisiones, de esta manera el cliente y el encargado del desarrollo negociarán un producto para que sea eficiente en términos de costo.

La Definición de Requisitos se refiere a la identificación de las funciones básicas del componente de programación en un sistema. Se pone atención en las funciones y restricciones bajo las cuales se deben de desarrollar.

La decisión de cómo se instrumentará la programación se retrasa hasta llegar a la fase de diseño. El producto de la Definición de Requisitos, es una especificación que describe el ambiente de procesamiento, las funciones requeridas de los programas, las restricciones de configuración sobre los programas (tamaño, velocidad, configuración de equipo), manejo de excepciones, subconjuntos y prioridades de instrumentación, cambios probables y modificaciones factibles, así como los criterios de aceptación del producto de programación.

2.2.3 Diseño del sistema

Después del Análisis en el modelo de fases, el Diseño de la programación es el siguiente paso. El Diseño de un sistema de información produce los detalles que establecen la forma en la que el sistema cumplirá con los requerimientos identificados durante la fase de análisis. Se refiere a la identificación de los componentes de la programación (funciones, flujo de datos y almacenamiento), especificando las relaciones entre ellos, la estructura de la programación, y manteniendo un registro de las decisiones, proporcionando un documento base para la instrumentación.

El Diseño se divide a su vez en estructural y detallado. Véase Figura 2.2.3.1.

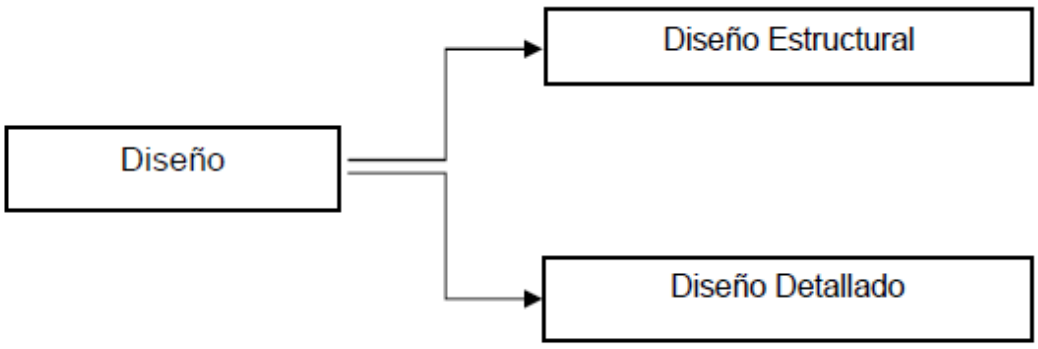

Figura 2.2.3.1 División de la fase de Diseño.

El diseño estructural comprende la identificación de los componentes de la programación, su desacoplamiento y descomposición en módulos de procesamiento y estructuras de datos conceptuales, así como la especificación de las interconexiones entre componentes.

El diseño detallado se refiere a detalles de: cómo empacar módulos de procesamiento, y cómo instrumentar los algoritmos, las estructuras de datos y sus interconexiones. Este diseño se relaciona con la adaptación de código existente, modificación de Algoritmos estándar, invención de nuevos algoritmos, diseño de representaciones de datos, e integración del producto final.

El Diseño detallado no es igual que la instrumentación. El primero está muy influido por el lenguaje de programación, pero no tiene que ver con aspectos sintácticos del mismo o con un nivel de detalle como evaluación de expresiones y estatutos de asignación.

2.2.4 Instrumentación

La fase de Instrumentación en el desarrollo del producto incluye la traducción de las especificaciones del diseño en código fuente, así como su depuración, documentación y pruebas. Los lenguajes de programación modernos proporcionan muchas características para mejorar la calidad del código fuente, como elementos estructurados, tipos de datos predefinidos o definidos por el usuario, verificación de tipos, reglas flexibles de cobertura, mecanismos para manejo de excepciones, elementos concurrentes, y módulos con compilación separada. Algunas de estas características se pueden simular en los lenguajes primitivos, mediante un estilo disciplinado de programación.

Los errores descubiertos durante la fase de instrumentación pueden ser errores en las interfaces de datos entre rutinas, errores lógicos en los algoritmos, errores en las

estructuras de datos y de falta de consideración de casos de procesamiento. Además, el código fuente puede contener: errores de requisitos, que indican alguna omisión en las necesidades del usuario en el documento de requisitos; errores de diseño, que reflejarán una mala traducción de requisitos en especificaciones y, por último, errores de instrumentación debido a una mala traducción de especificaciones en código fuente.

Una de las metas principales del modelo de fases para el desarrollo de productos de programación es la eliminación de errores de requisitos y diseño, antes de iniciada la instrumentación debido a que es muy caro eliminar errores del análisis y el diseño del código fuente durante la instrumentación y las pruebas.

2.2.5 Pruebas

Las pruebas del sistema comprenden dos tipos de actividades: pruebas de integración y de aceptación. El desarrollo de una estrategia para integrar los componentes de un sistema de programación, en una unidad funcional, requiere una planeación cuidadosa de modo que se disponga de los módulos cuando éstos se necesiten. Las pruebas de aceptación se relacionan con la planeación y ejecución de varios tipos de pruebas para demostrar que el sistema de programación instrumentado satisface las +necesidades establecidas en el documento de requisitos.

2.2.6 Implementación

Una vez aceptado por el cliente, el sistema de programación se entrega para su Implementación y operación. Se implementan los niveles de hardware y software que componen el proyecto y el sistema creado en particular. Es en esta etapa cuando realmente se pone a prueba el sistema, para forzarlo y obtener todo el rendimiento posible.

2.2.7 Mantenimiento

Para la fase de Mantenimiento se incluyen actividades como mejoras de las capacidades, adaptación a nuevos ambientes de procesamiento y corrección de fallas del sistema.

2.2.8 Ventajas y desventajas del Método de Ciclo de vida clásico.

Ventajas:

- Este modelo es el más ampliamente usado por los ingenieros.
- Es muy útil pues ayuda a los desarrolladores a comprender qué es lo que tienen que hacer en cada momento. Su simplicidad hace que resulte sencillo explicárselo a los clientes que no están familiarizados en el proceso software. Además, se muestran de forma explícita qué productos intermedios se tienen que obtener antes de abordar las siguientes tareas.
- El riesgo es bajo para los casos en que se conocen muy bien los requerimientos desde el inicio.
- Es fácil de aplicar.
- Es un modelo sencillo y disciplinado.
- Es fácil aprender a utilizarlo y comprender su funcionamiento.
- Está dirigido por los tipos de documentos y resultados que deben obtenerse al final de cada etapa
- Ha sido muy usado y, por tanto, está ampliamente contrastado.
- Ayuda a detectar errores en las primeras etapas a bajo costo.
- Ayuda a minimizar los gastos de planificación, pues se realiza sin problemas.

Este modelo lineal secuencial es el paradigma más antiguo y más extensamente utilizado en la ingeniería del software. Sin embargo, la crítica del paradigma ha puesto en duda su eficacia.

Desventajas

Las desventajas que presenta el Modelo de Ciclo de Vida Clásico son:

- Los proyectos raramente siguen el proceso lineal tal como se definía originalmente el ciclo de vida.
- Es difícil que el cliente exponga explícitamente todos los requisitos desde el principio.
- El cliente debe tener paciencia. Una versión de trabajo del programa no estará disponible hasta que el proyecto esté muy avanzado. Un grave error puede traer problemas si no se detecta hasta que se revisa el programa.
- Puede resultar complicado regresar a etapas anteriores (ya acabadas) para realizar las correcciones.
- El producto final obtenido puede que no refleje todos los requisitos del usuario.

- Los proyectos reales raras veces siguen la secuencia que propone el modelo. Aunque el modelo lineal puede acoplar interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el equipo del proyecto comienza.

- Se tienen dificultades para hacer cambios en cualquiera de las fases del ciclo.
- La versión funcionando del sistema no estará disponible hasta las etapas finales del desarrollo del proyecto.
- El tiempo que se pasa esperando puede sobrepasar el tiempo que se emplea en el trabajo productivo. Los estados de bloqueo tienden a ser más importantes al principio y al final de un proceso lineal secuencial.

Cada uno de estos errores es real. Sin embargo, el paradigma del ciclo de vida clásico tiene un lugar definido e importante en el trabajo de la ingeniería del software.

A pesar de su antigüedad, el ciclo de vida clásico se ha ganado un lugar importante en el área de la Ingeniería del Software. Proporciona una guía de trabajo en la que se encuentran métodos para el análisis, diseño, codificación, pruebas y mantenimiento. El ciclo de vida en cascada sigue siendo el modelo de proceso más extensamente utilizado por los ingenieros del software, principalmente por su sencillez y facilidad de llevar a cabo. Pese a tener debilidades, es significativamente mejor que un enfoque arbitrario (como el de codificar y corregir) para el desarrollo del software. Muchos de los posteriores modelos de ciclo de vida son, en realidad, modificaciones sobre el modelo clásico, al que se le incorporan nuevas actividades.

2.3 CARACTERISTICAS, VENTAJAS Y DESVENTAJAS DE MICROSOFT ACCESS.

Microsoft Access es un **Data Base Management System (Sistema de Administración de Bases de Datos, DBMS)** clasificado como producto de escritorio, de uso personal o de pequeñas empresas. Es un componente del paquete Microsoft Office. Su función es ser una herramienta de desarrollo de base de datos, capaz de trabajar en si misma o bien con conexión hacia otros lenguajes de programación, tales como Visual Basic 6.0 (VB6) o VisualBasic.NET. Pueden realizarse consultas directas a las tablas contenidas mediante instrucciones SQL. Internamente trae consigo el lenguaje Visual Basic for Application (VBA) el cual es similar en forma a VB6.

De las herramientas rivales del Microsoft Office: IBM Lotus Symphony, Libre Office, Google Docs, Zoho y SoftMaker; solo OpenOffice cuenta con un DBMS, por lo que el Microsoft Access, bien puede decirse, no tiene competencia en el mercado.

2.3.1 Ventajas de Microsoft Access

Quizá una de las mayores ventajas que ofrece el Microsoft Access es su capacidad para vincularse con las otras herramientas de la suite Office. Esto permite al usuario interactuar con Word, Excel, PowerPoint y Outlook ya sea recibiendo, enviando información entre ellas. En otras palabras permite el desarrollo de aplicaciones basadas en productos Microsoft. En la figura 2.3.1.1 podemos observar la presentación comercial de Microsoft Access.

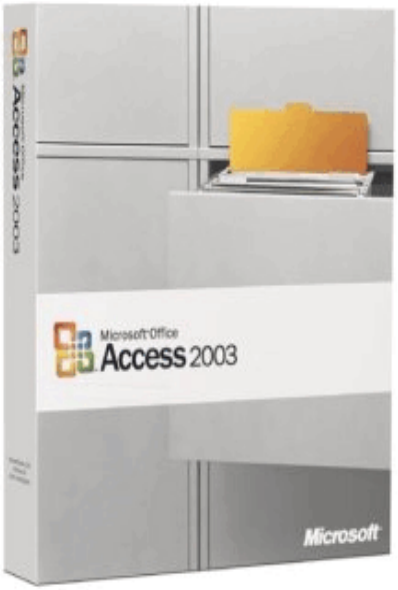

Figura 2.3.1.1 Microsoft Access

Pero dichas facilidades se extienden a otros productos de Microsoft, como: Visual Basic, MySql y SQLserver, lo que lo hace más versátil.

Su funcionamiento se basa en un motor llamado Microsoft Jet y permite el desarrollo de pequeñas aplicaciones autónomas formadas por formas Windows y código VBA. Una posibilidad adicional es la de crear archivos con bases de datos que pueden ser consultados por otros programas.

Entre las principales funcionalidades de Microsoft Access se encuentran:

- Crear tablas de datos indexadas.
- Modificar tablas de datos.
- Crear bases de datos relacionales mediante las relaciones entre tablas
- Creación de consultas y vistas.
- Consultas por referencias cruzadas.
- Modificación de bases de datos.
- Formularios.

- Informes.
- Llamadas a la **Aplication Programation Interface (Interfaz de Programación de Aplicaciones, API)** de Windows.
- Interacción con otras aplicaciones que usen VBA (resto de aplicaciones de Microsoft Office, Autocad, etc.).
- Soporte Unicode: El soporte Unicode de Access permite a las organizaciones multinacionales trabajar con versiones de la aplicación en lenguas diferentes.
- Posibilidad de guardar los archivos en formatos de Access anteriores.
- Autocorrección de nombres: resuelve automáticamente el efecto secundario producido al cambiar el nombre de un objeto de la base de datos.
- Formato condicional: permite utilizar números negativos y positivos, además de valores que pueden expresarse como menor que, mayor que, entre, o igual a.
- Asistente para imprimir tablas relacionadas.
- Agrupación de controles en una sola unidad.
- Compresión automática de la base de datos al cerrar el archivo si la reducción del espacio en disco es significativa.
- Páginas de acceso a datos agrupados, permite a los usuarios extender las aplicaciones de bases de datos a una Intranet corporativa.
- La lista de campos permite agregar información a una vista Página de acceso a datos con sólo arrastrar y colocar los campos.

A continuación se listan algunas de las características técnicas de Microsoft Access, publicadas por el fabricante y que evidencian sus alcances:

- Tamaño del archivo de la base de datos Access: 2GB menos el espacio necesario para los objetos del sistema. (Nota: El tamaño máximo en un archivo sencillo de bases de datos es de 2GB. Se puede trabajar sobre esta limitación enlazándose a tablas de otras bases de datos Access, cada una de las cuales puede ser de hasta 2GB).
- Número de Objetos en una base de datos: 32,768.

- Número de Módulos (Incluyendo formas y reportes que tengan la propiedad HAS Module en Verdadero (True): 1000.
- Número de caracteres en el nombre de un objeto: 64.
- Número de caracteres en un password: 20.
- Número caracteres en un nombre de usuario o grupo: 20.
- Número de usuarios concurrentes: 255.
- Número de tablas abiertas: 2048.
- Tamaño de la tabla: 2GB.
- Número de índices en una tabla: 32.
- Número de campos indexados: 10.
- Tamaño de un campo objeto OLE: 1 GB.

2.3.2 Desventajas de Microsoft Access

A continuación se presentan algunas de las desventajas más notables del Microsoft Access:

- Problemas a la hora de importar archivos de aplicaciones distintas de Office.
- No es multiplataforma, pues sólo está disponible para sistemas operativos de Microsoft y que no permite transacciones de información con facilidad.
- Para bases de datos de gran tamaño (en cuanto a volumen de datos o de usuarios) es recomendable usar otros sistemas como Microsoft SQL Server, MySQL, Oracle, Informix SQL, ya que no están limitados a un archivo máximo de 2GB.

2.4 CARACTERÍSTICAS, VENTAJAS Y DESVENTAJAS DE VISUAL BASIC 6.

2.4.1 Origen de Visual Basic.

Visual Basic es un lenguaje de programación dirigido por eventos, este lenguaje de programación es un dialecto de **BASIC (Beginner's All purpose Symbolic Instruction Code (Todo el objetivo del Principiante, Código de Instrucción Simbólico)**, con importantes agregados. Su primera versión fue presentada en 1991, con la intención de simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilitara la creación de interfaces gráficas y en cierta medida, también la programación misma.

Es un lenguaje de fácil aprendizaje, guiado por eventos, y centrado en un motor de formularios poderoso que facilita el rápido desarrollo de aplicaciones gráficas. Su principal innovación, que luego fue adoptada por otros lenguajes, fue el uso de un tipo de **DLL (Biblioteca de Enlace Dinámico)**, llamado inicialmente **VBX(Extensión de Visual Basic)** y posteriormente **OCX(Objeto de Vinculación e Integración)**, que permiten contener toda la funcionalidad de un control y facilitar su rápida incorporación a los formularios. Véase figura 2.4.1.1

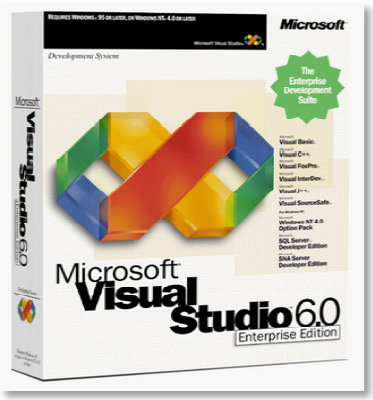

Figura. 2.4.1.1 Paquetería de Microsoft Visual Studio 6.0 (Visual Basic).

2.4.2 Características Generales

Las aplicaciones creadas con **Visual Basic** se desarrollan partiendo de una interfaz gráfica con la cual es posible generar, de manera automática, conectividad entre controles y datos de manera fácil y rápida mediante la acción de arrastrar y colocar objetos sobre formularios. Los programas quedan compuestos, en una parte, por códigos y, en otra, por partes asociadas a los objetos presentes en la interfaz gráfica. Esto hace que el tipo de programación propio de **Visual Basic** sea parcialmente estructurado y parcialmente orientado a objetos.

Visual Basic es un ambiente de programación integrado, esto quiere decir que se utiliza la misma herramienta para editar, compilar, correr, etc. una aplicación. Los procesos se componen de formularios y controles, los cuales tienen propiedades y están diseñados para llevar a cabo procedimientos cuando ocurre algún evento sobre ellos. Además el programa permite usar procedimientos reutilizables que pueden haber sido creados por otras personas o por uno mismo así como módulos de clase para programación orientada a objetos.

Visual Basic puede crear componentes **COM (Módulo de Objeto Componente)**, los cuales constituyen la estructura que utiliza Microsoft para todos sus lenguajes actualmente, para desarrollar aplicaciones se utilizan componentes y objetos, parte de esos componentes son los controles ActiveX, servidores Active (DLL o EXE), documentos ActiveX, etc. Cuenta con características para el desarrollo de Internet e Intranet como lo son las aplicaciones **DHTML (Lenguaje Dinámico de Marcado de Hipertexto)**, aplicaciones **IIS (Información de Servicios de Internet)** y soporte para el **FTP (Protocolo de Transferencia de Archivos).**

La creación de un programa bajo **Visual Basic** requiere de los siguientes pasos:

- La creación de una interfaz de usuario. Esta interfaz constituirá la vía de comunicación con la máquina para la entrada y salida de datos. Es parte de una ventana o formulario a la que le se le van añadiendo los controles pertinentes.
- La definición de las propiedades de los controles u objetos que hayan sido colocados en el formulario. Estas propiedades determinan la forma estática de los controles, es decir, su forma y sus funciones.
- La generación de un código asociado a los eventos que ocurran sobre los objetos. A la respuesta a un evento (clic, doble clic, una tecla pulsada, etc.) se le llama procedimiento y deberá desarrollarse de acuerdo al propósito del programa.
- La generación del código del programa. Un programa puede elaborarse simplemente con la programación de los procedimientos que deban acompañar a cada evento sobre un objeto; sin embargo, **Visual Basic** ofrece la posibilidad de establecer un código de programa separado de estos eventos. Este código puede introducirse en módulos, funciones y procedimientos. Estos procedimientos no responden a un evento acaecido a un objeto, sino que responden a un evento producido durante la ejecución del programa. Visual Basic es extensible gracias a las llamadas al **API (Interfaz de Programación de Aplicaciones)** de Windows, al soporte de terceras partes y a la integración con otras aplicaciones de Windows. Es utilizado principalmente para aplicaciones de gestión de empresas, debido a la rapidez con la que puede hacerse un programa que utilice una base de datos sencilla, dado que cuenta con varias bibliotecas para el manejo y acceso a bases de datos, entre las que destaca ADO, Jet, ODBC y RDO; programadores en este lenguaje.
- Finalmente, las numerosas bibliotecas **DLL (Biblioteca de Enlace Dinámico)** facilitan el acceso a muchas funciones del sistema operativo y la integración con otras aplicaciones. **Visual Basic** tiene también sus lenguajes derivados como lo son el VBScript y el Visual Basic for Applications VBA, el primero es el lenguaje predeterminado para Active Server Pages ASP y el segundo es más bien una

extensión del mismo lenguaje, el cual permite codificar módulos o macros que se utilizan en los programas de Microsoft Office.

En la actualidad existe una nueva variante del lenguaje que se llama Visual Basic.NET, el cual es un programa que cuenta ya con funciones equivalentes a C++ y con una orientación a objetos como la de JAVA.

2.4.3 Ventajas y Desventajas de Visual Basic.

Ventajas

- Visual Basic es un lenguaje simple y, por tanto, más fácil de aprender que otros lenguajes más complejos como DELPHI y Power Builder, entre otros. Su mayor simplicidad radica en el desarrollo de formularios, mediante el arrastre de controles. Es un lenguaje compatible con Microsoft Office y otros lenguajes (Informix, Oracle, Microsoft SQL, etc).
- Gran parte del trabajo en el diseño de formularios está realizado, gracias a la gran gama de controles incorporados junto al lenguaje, cuyas propiedades y métodos son fáciles de manipular, lo cual, junto con todas las demás características del programa, hace que el desarrollo de aplicaciones se vuelva sumamente rápido, permite crear controles personalizados fácilmente del mismo modo que el diseño de los formularios.
- Puede crear controles ActiveX más fácilmente que si se usa C++, cuenta con un excelente paquete de ayuda, cuenta con herramientas para el desarrollo integración de Internet/Intranet, se pueden crear servidores fácilmente. Es excelente para cálculos intensivos del **CPU (Unidad Central de Procesos)** como por ejemplo operaciones matemáticas.

Desventajas

- El dueño de Visual Basic es Microsoft, por lo tanto nadie que no sea del equipo de desarrollo de esta compañía decide la evolución del lenguaje.

- Sólo existe un compilador, llamado igual que el lenguaje, generando ejecutables para Windows, la sintaxis es bastante inflexible, los ejecutables generados son relativamente lentos, no es adecuado para aplicaciones grandes, multimedia, de oficina, videojuegos, editores gráficos, etc.

- No cuenta con características para programación avanzada, no permite generar librerías dinámicas (DLL), sólo permite el uso de funciones de librerías dinámicas (DLL) stdcall. para que los ejecutables que generan funcionen, necesita una DLL llamada MSVBVMxy.DLL: MicroSoft Visual Basic Virtual Machine x.y.

- Algunas funcionalidades están indocumentadas, la ligera implementación de la OOP no permite sacar el máximo provecho de este modelo de programación.

- No soporta el tratamiento de procesos como parte del lenguaje, no maneja excepciones, no incluye operadores a nivel de bits, no contempla el manejo de memoria dinámica, punteros, arrays, etc. como parte del lenguaje.

- No puede avisar ni advertir cuando están presentes ciertos errores, como sería una inadecuada conversión de tipos.

- El tratamiento de mensajes de Windows es básico e indirecto; la gran gama de controles incorporados son, sin embargo, muy generales, lo cual lleva a tener que reprogramar nuevos controles para una necesidad particular de la aplicación.

- Los controles personalizados no mejoran la potencia de la API de Windows y, en algunos casos, acudir a ésta es la única manera de conseguir el control personalizado deseado.
- No tiene la misma funcionalidad que C++ a la hora de obtener características de bajo nivel del sistema operativo. La forma de programación que plantea Visual Basic ha ocasionado que muchos programadores de este lenguaje practiquen malos hábitos, entre los que se encuentran:

  - La utilización de variables globales.
  - Omitir la declaración de variables.
  - Trabajar con variables de tipo indefinido (Tipo Variant).
  - Escribir código innecesario.
  - Escribir código repetido a falta de funciones y procedimientos.
  - El uso incorrecto de la API de Windows.
  - El uso de goto y etiquetas.
  - El uso de controles como simples contenedores de datos.
  - El crear dependencia de los controles a la hora de programar.

2.5 CARACTERÍSTICAS, VENTAJAS Y DESVENTAJAS DE LA ARQUITECTURA CLIENTE-SERVIDOR.

La arquitectura cliente-servidor es una técnica para dividir y especializar programas y equipos de cómputo, a fin de que la tarea que cada uno de ellos realiza se efectúe con la mayor eficiencia y permita simplificarlas.

En esta arquitectura la capacidad de proceso está repartida entre el servidor y los clientes. Los clientes interactúan con el usuario, usualmente en forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Los servidores proporcionan un servicio al cliente y devuelven los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Además deben manejar los interbloqueos, la recuperación ante fallas, y otros aspectos afines.

Para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, que proporcione los mecanismos básicos de direccionamiento y transporte.

2.5.1 Características del Modelo Cliente-Servidor.

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas o en la misma plataforma.

- Un servidor da servicio a múltiples clientes.
- Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

El Cliente actúa de la siguiente forma:

- Establece una conexión con el servidor (Crea un socket con el servidor).
- Manda mensajes al servidor o espera un mensaje de él (Consultas).
- Repite el paso anterior mientras sea necesario.
- Cerrar la conexión con el servidor.

El Servidor actúa de la siguiente manera:

- Inicializa un puerto de comunicación, en espera de clientes que intenten conectarse a él (Crea un serverSocket).
- Una vez que se conecta alguien, crea un **thread (hilo)** de ejecución para este usuario mientras que el thread principal vuelve al paso anterior. Esto comúnmente se hace para que el servidor pueda atender a varios clientes al mismo tiempo.
- Se comunica con el cliente mediante el socket creado entre el cliente y él.
- Espera que el cliente se vaya o lo desconecta el mismo servidor (Cierra el socket entre ellos) y elimina el thread de comunicación entre ellos.

2.5.2 Clasificación por capas de la arquitectura Cliente-Servidor.

2.5.2.1 Modelo Cliente-Servidor de una capa.

Las aplicaciones cliente-servidor de una capa agrupan la lógica de presentación (interfaz de usuario), la lógica de aplicación y la fuente de datos (base de datos) en una sola máquina, la cual actúa simultáneamente como cliente y servidor.

2.5.2.2 Modelo Cliente-Servidor de dos capas.

Tradicionalmente la arquitectura Cliente-Servidor está basada en un modelo de computación de dos capas. Este modelo consiste de un cliente y un servidor de base de datos. El procesamiento de tareas y la lógica de la aplicación son compartidas entre el servidor de base de datos y el cliente.

Como podemos observar en la figura 2.5.2.2.1, en este modelo a los clientes se les llama clientes pesados, en donde reside mucho del poder de procesamiento y de la lógica de la aplicación. Esto hace que el mantenimiento del cliente sea costoso. Adicionalmente, los clientes pueden estar operando en diferentes plataformas, dando como resultado una distribución compleja de versiones de aplicaciones específicas de las plataformas.

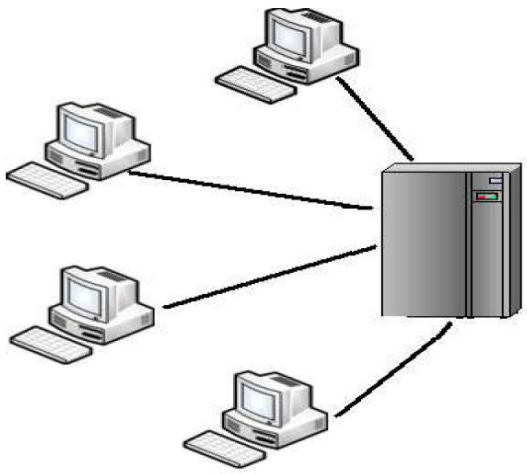

Figura 2.5.2.2.1 – Arquitectura Cliente Servidor de 2 capas

2.5.2.3 Modelo Cliente-Servidor de tres capas.

El modelo de computación de tres capas es un modelo evolucionado del modelo de dos capas. En este modelo, existe una capa intermedia entre el cliente y el servidor de base de datos. Esta capa consiste de un servidor de aplicaciones que contiene el grueso de

la lógica de la aplicación. Los clientes en este modelo son clientes livianos o clientes ligeros. Con esta arquitectura la lógica de la aplicación reside en una sola capa que puede ser fácilmente mantenida. El diseño arquitectónico de la capa media puede también ser optimizada en funciones del servidor puesto que éste no tiene que contener u hospedar la base de datos.

En esta arquitectura de tres capas, el software del cliente (capa cliente) es ligero, suficiente para ser descargado bajo demanda y lo suficientemente pequeño como para presentar la interfaz del usuario. El grueso de la lógica de la aplicación está implementada ya sea en la capa media (servidor de aplicaciones) o está almacenada en la base de datos, como podemos observar en la figura 2.5.2.3.1.

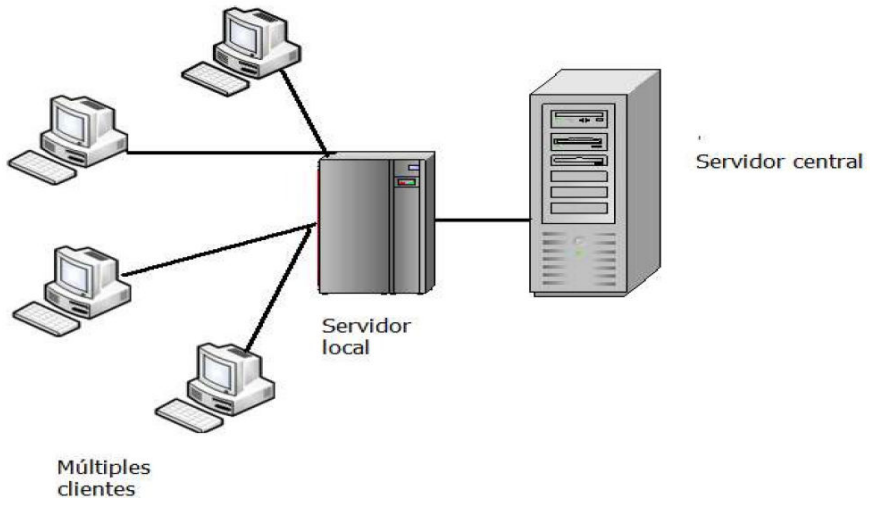

Figura 2.5.2.3.1 – Arquitectura Cliente Servidor de 3 capas

2.5.2.4 Modelo Cliente-Servidor de n capas.

El despliegue de una plataforma capaz de soportar los requerimientos de aplicaciones en Internet, invariablemente requieren de una arquitectura de alto nivel de múltiples capas.

En la capa cliente, los clientes con un navegador basado en HTML soportan los requerimientos de accesibilidad **anytime (cuando sea)** y **anywhere (donde sea)**: **GUIs (Graphical User Interface, Interfaz grafica de usuario)** más complejas, pueden ser soportados dentro del navegador descargando más código de la aplicación cliente, descansando en **plugins** (**Accesorios**) o en capacidades específicas del navegador.

La capa del servidor de presentación consiste de servidores WEB que manejan requerimientos http y genera dinámicamente código de presentación para su ejecución y despliegue en el cliente, de forma típica en la forma de páginas HTML. Muchos de los requerimientos de personalización son manejados en esta capa. El servidor de presentación puede parametrizar la generación de código de presentación del cliente basándose en el **ID** (**Identifier –Identificador**) del usuario, preferencias, roles, afiliación, etc.

Los servidores dan cabida a la lógica de la aplicación que idealmente puede ser reutilizada a través de una variedad de clientes y otras aplicaciones. La lógica de la aplicación también debería poder ser llamada fácilmente desde clientes y servidores externos para facilitar los requerimientos de integración de ésta. La capa del servicio de la arquitectura también contiene servidores especializados de análisis y reportes para complementar la aplicación.

Finalmente, la capa de servidores de datos converge hacia una implementación estándar suficiente para satisfacer las demandas de almacenamiento, manipulación, recuperación y análisis de requerimientos de aplicaciones en Internet. Esta capa

también contiene fuentes de datos externas y aplicaciones que deben estar integradas en el sistema. En la figura 2.5.2.4.1 podemos observar una representación de la interacción lógica de este modelo de la arquitectura.

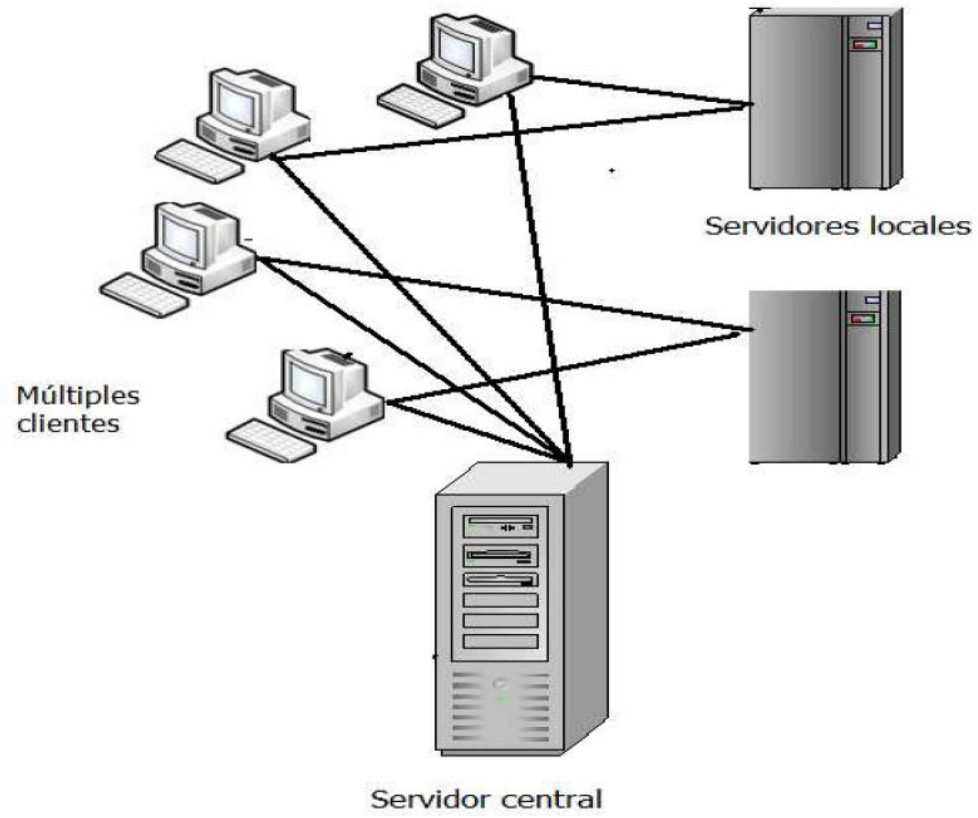

Figura 2.5.2.4.1 - Arquitectura Cliente Servidor de n capas

2.5.3 Elementos de red dentro de la arquitectura Cliente-Servidor.

2.5.3.1 Hub

Un **Hub** (**Concentrador**) es un equipo de conexión de redes, que funciona como repetidor de la señal eléctrica. Es decir toma la señal que le llega por un puerto y la reproduce íntegramente por los demás. Nada de comprobación de tramas ni toma de decisiones. El ancho de banda es compartido por todos los equipos conectados a él.

2.5.3.2 Bridge

Un **bridge** (**Puente**) es un dispositivo de interconexión de redes que opera en la capa 2 (nivel de enlace de datos) del modelo OSI. Este interconecta dos segmentos de red (o divide una red en segmentos) estableciendo el paso de datos de una red hacia otra, con base en la dirección física de destino de cada paquete.

Funciona a través de una tabla de direcciones **MAC** (**Media Access Control- Control de Acceso al Medio**) detectadas en cada segmento a que está conectado. Cuando detecta que un nodo de uno de los segmentos está intentando transmitir datos a un nodo del otro, el bridge copia la trama para la otra subred.

La principal diferencia entre un bridge y un hub es que el segundo pasa cualquier trama con cualquier destino para todos los otros nodos conectados, en cambio el primero sólo pasa las tramas pertenecientes a cada segmento. Esta característica mejora el rendimiento de las redes al disminuir el tráfico inútil.

2.5.3.3 Switch

El **switch** (**Conmutador**) es un dispositivo muy similar al bridge, opera en la misma capa del modelo OSI y tiene un comportamiento similar, con la diferencia de enlazar más de 2 segmentos de red, pueden ser 3 o más. Opera en forma similar, mediante tablas de dirección MAC, en donde cada puerto contiene una lista de las direcciones de destino que son alcanzables dentro del mismo.

2.5.3.4 Router

Un **router** (**Ruetador**) es un dispositivo hardware o software (vrouter –virtual router) de interconexión de redes de computadoras que opera en la capa tres (nivel de red) del

modelo OSI. Este dispositivo interconecta segmentos de red o redes enteras. Hace pasar paquetes de datos entre redes tomando como base la información de la capa de red.

El router toma decisiones lógicas con respecto a la mejor ruta para el envío de datos a través de una red interconectada y luego dirige los paquetes hacia el segmento y el puerto de salida adecuados. Sus decisiones se basan en diversos parámetros. Una de los más importantes es decidir la dirección de la red hacia la que va destinado el paquete (En el caso del protocolo **IP** (**Internet Protocol**), esta sería la dirección IP). Otras decisiones son la carga de tráfico de red en las distintas interfaces de red del router y establecer la velocidad de cada uno de ellos, dependiendo del protocolo que se utilice.