Universidad Nacional Autónoma de México

F A C U L T A D D E I N G E N I E R Í A

Sistema de Calificación y Seguimiento de Pacientes del Centro ACASULCO

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero en Computación

P R E S E N T A

Apolo Isaac Hernández Ávila

Hugo Camacho Pérez

Director: Ing. Carlos Alberto Román Zamitiz

Ciudad Universitaria, 2009.

Quiero dedicar este trabajo a mis padres José Camacho Palacios y Rafaela Pérez Piedra por haberme dado la oportunidad de estudiar y contar con su apoyo incondicional en todo momento, sin sus valores, consejos y enseñanzas hubiera sido imposible obtener este resultado.

A las futuras generaciones de mi familia: Rafael y Valeria, que lograrán esto y más.

A mi primo Miguel A. Pérez Camacho por sus consejos y la confianza que siempre ha depositado en mí.

A mis amigos con los que he compartido excelentes momentos y por lo que hemos aprendido juntos, en especial a Vicente Flores Mojica †.

A todas las personas con quien he compartido esta vida y cuyo presente trabajo no puedo entender sino como el resultado de cada granito de arena que han dejado en mí.

Gracias a la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por haberme permitido ocupar un lugar en sus aulas, a las excelentes personas que son los profesores de esta Facultad y al Ing. Carlos Román Zamitiz por dirigir esta tesis.

Hugo Camacho Pérez

Esta tesis la dedico a mis padres Héctor y Amparo y a mis hermanos Jocelyn y Héctor quienes han sido mi principal apoyo durante toda mi vida, ya que sus consejos, comprensión y amor fueron la motivación para concluir con mis estudios profesionales. Igualmente se la dedico a mis tíos María de los Ángeles, Agustín, Nancy y Gabriel quienes han estado conmigo en todo momento y son una parte muy importante de mi vida. Se la dedico a mi novia Ely que con su amor, apoyo y compañía me ha motivado a lograr todo lo que me proponga, a mis suegros Roberto y Juana y a mi cuñada Blanca que han sido como mi segunda familia. También le dedico este trabajo a mis amigos: Hugo, Logan, Juanito, Ana, Chars, Xavier, Isaac, Enrique, Pancho, Café, Chucho, Popotes, Estephanie, Gude, Andy, Chenge, Ángel, Miguel, Omar, Chuchin, Alberto, Alma, Israel, Carlos, Vero, Celia, Araceli, Alejandra, Roy, Elsa, Oswaldo y a todos que me faltaron en esta larga lista, les agradezco mucho su amistad y aquellos momentos que he compartido con ustedes. Finalmente quiero dedicar y agradecer a todos mis profesores que me han dejado más que solo preparación académica, dándome lecciones de vida, consejos y apoyo. Gracias a todos ustedes me he convertido en la persona que soy y quiero compartirles la alegría de ser un orgulloso ingeniero egresado de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

Apolo Isaac Hernández Ávila

1. INTRODUCCIÓN

Hoy en día, hemos sido testigos de que los avances tecnológicos han aportado grandes ventajas en cualquiera de los campos donde han sido aplicados, es por ello que la implantación de los sistemas computacionales han ayudado a automatizar tareas y aligerar el tiempo de trabajo del hombre y así hacer más productivo su trabajo.

Las soluciones en sistemas computacionales hechas a la medida, dotan a una organización de una mejor producción y desempeño de sus actividades en todas sus áreas. En empresas u organizaciones de servicio, utilizar sistemas de información mejora sustancialmente el rendimiento en el servicio al cliente.

Uno de los principales temas que aquejan a nuestra sociedad, es que se ha incrementado el uso de sustancias como el alcohol y las drogas. Este problema es uno de los principales culpables del sufrimiento de consumidores que lo llegan a perder todo incluso sus propias vidas ya que no existe el apoyo suficiente que les permita salir de esta situación. Hoy en día existe un centro de atención psicológica llamado ACASULCO el cual da sus servicios aplicando un programa llamado “Programa de Satisfactores Cotidianos” el cual ha ayudado a varias personas a controlarse por sí mismos hasta dejar por completo el uso de sustancias nocivas. Actualmente el centro aplica pruebas en cada una de las etapas del programa, realizándolas de manera manual y no cuenta con un expediente controlado lo cual afecta directamente el seguimiento de pacientes influyendo de manera negativa en la efectividad de dicho programa.

Por tales motivos, se propone la creación de un sistema que permita llevar el control de los pacientes, apoyando en la aplicación y calificación de pruebas psicológicas, y que además conjunte un expediente de los mismos.

Otro de los puntos que se deben tomar en cuenta, es que se ha elegido implementar un sistema Web que aproveche las ventajas del Internet. Con ello se pretende dar apoyo a pacientes que no puedan asistir al centro para aplicarle su prueba psicológica, además de que podrán revisar su calendario de citas y su avance en el programa. El beneficio también es para los terapeutas que podrán gestionar las sesiones con sus pacientes, subir a sus expedientes información importante y programar pruebas que les den certeza de su avance.

Para lograr el objetivo, se desarrollaran en el primer capítulo las bases teóricas necesarias para el desarrollo de software, tales como Bases de Datos, Ingeniería de Software, Orientación a Objetos, UML, etc.

En el segundo capítulo, veremos la obtención de requerimientos, su importancia y aplicación, siendo esta una herramienta que nos ayude a conocer las necesidades de los usuarios.

El tercer capítulo, tratará del análisis de los requerimientos, de los diagramas necesarios para el claro entendimiento del problema y así tener una visión más clara del software que solucionará a cada requerimiento.

Enseguida, en el cuarto capítulo se llevará a cabo el diseño del sistema, donde se profundizará en temas como la elección de la tecnología al hacer un comparativo de las ventajas y desventajas de su uso y así elegir la mejor.

En el quinto capítulo, se verá el desarrollo del producto final, así como la aplicación de mejoras o mantenimientos detectados durante el ciclo de desarrollo de software.

Finalmente en el sexto capítulo se tratará el tema de pruebas y liberación del sistema.

2. ANTECEDENTES

2.1.La importancia del Software.

Software y hardware son dos conceptos que hemos asociado a la modernidad, sin embargo son en realidad nombres nuevos de prácticas ancestrales. Las culturas antiguas desarrollaron diferentes técnicas de conteo y cálculo con sus correspondientes instrumentos y notaciones, por citar un ejemplo tenemos el ábaco que se ha encontrado en varias culturas con diferentes variaciones. A los instrumentos hoy en día los denominamos como hardware y la habilidad de usarlos o interpretarlos constituye el software.

De esta manera aunque la palabra software no nace con los equipos electrónicos, si es con ella que adopta el nombre. Sin embargo, en estos instrumentos de las culturas antiguas, el software no se encuentra incorporado a los instrumentos sino que es aportado por el operario.

En la actualidad la definición más formal de software es la siguiente:

"Es el conjunto de los programas de cómputo, procedimiento, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación."

Con esta definición, el concepto de software va mas allá de lógica e interpretación que se pueda obtener con el hardware, abarca también la lógica en sus diferentes estados, la documentación, los datos a procesar y la información que el usuario aporta forma parte del software.

2.1.1. Clasificación del software.

La importancia del software fue creciendo paulatinamente. En un principio la proporción favorecía al equipo físico, pero progresivamente la importancia del software creció, hasta llegar a un grado en convertirse en el más importante. Podemos clasificar el software de acuerdo al uso que le damos de manera general, en tres grandes rubros:

- Lenguajes de programación.
- Sistemas Operativos.
- Programas de aplicación.

2.1.2. Lenguajes de programación.

Un lenguaje de programación es un conjunto de signos y reglas que permite la comunicación con un ordenador o computadora. De acuerdo a la forma de operación de los lenguajes de

programación podemos clasificarlos en ensambladores, interpretadores, compiladores e híbridos (interpretadores y compiladores).

Los lenguajes ensamblador ofrecen al programador un modelo muy riguroso de la máquina, ya que dispone de todos los recursos en su nivel más elemental. La traducción del programa se limita a una correspondencia entre un mnemónico y las instrucciones de máquina.

Los interpretadores utilizan símbolos, pero en este caso cada símbolo tiene una función asociada por el intérprete. El intérprete ejecuta cada una las instrucciones que encuentra en la estructura del programa en forma secuencial, detectando errores solo en el momento de ejecución.

Los compiladores realizan la traducción de los programas de un lenguaje a otro, generalmente se manejan dos tipos de archivos, el archivo fuente y el archivo destino, que en principio son diferentes, el archivo destino comúnmente es el lenguaje de la maquina o ensamblador.

Existen además leguajes de programación que podemos clasificar como compiladores e intérpretes (híbridos), debido a que incorporan en su ejecución una fase de compilación y de interpretación.

2.1.3. Sistemas Operativos

Los sistemas operativos surgen como extensiones de los lenguajes de programación. Los sistemas operativos son un conjunto de programas destinados a la administración de los recursos disponibles en una computadora, como es el uso de la memoria, el procesador, el acceso a dispositivos periféricos, etc.

Un sistema operativo se caracteriza por la forma en que efectúa ciertas tareas, como son:

Administración de tareas:

- Monotarea: Solo ejecuta un proceso en un momento dado. Una vez que ejecuta un procedimiento continuará haciéndolo hasta su finalización o interrupción.
- Multitarea: Ejecuta varios procesos al mismo tiempo. Este tipo de Sistemas Operativos asigna los recursos disponibles, de forma alternada a los procesos que lo solicitan, de tal forma que el usuario percibe que todos funcionan a la vez de forma concurrente.

Administración de usuarios:

- Monousuario: Solo permite ejecutar los programas de un usuario al mismo tiempo.
- Multiusuario: Permite que varios usuarios ejecuten simultáneamente sus programas, accediendo a la vez a los recursos de la computadora.

Administración de recursos.

- Centralizada: Si permite utilizar los recursos de una sola computadora.
- Distribuido: Si permite utilizar los recursos de más de una computadora al mismo tiempo.

2.1.4. Programas de Aplicación

Los programas de aplicación justificaron la construcción de equipos, inicialmente se construyeron aplicaciones militares, conforme se inicio la comercialización de las computadoras surgieron aplicaciones destinadas a la administración pública, tales como sistemas de nomina, sistemas de conteo, de inventario, estadísticas etc.

2.2.Ingeniería de Software.

En el nacimiento del desarrollo de software (1945 - 1955), programar no fue una tarea fácil, la programación estaba ligada al diseño y componentes de cada máquina, esto aunado a la complejidad de manejar el lenguaje ensamblador, dio inició a un periodo de crisis en el desarrollo del software.

Con la aparición de nuevos lenguajes de programación se realizaron innumerables sistemas que permitían la solución a problemas concretos pero que eran incapaces de comunicarse o compartir información entre si. Esto ocasiono el desarrollo inacabable de grandes programas que permitiera la gestión de toda la información en un mismo sistema, sin embargo desarrollar estos sistemas implicó un costo impredecible, ineficiencia, errores y un mantenimiento (cambios en los sistemas) que requería un costo tan alto como la creación del sistema mismo.

Bajo tales efectos, se buscaron técnicas innovadoras que permitirán realizar un profundo análisis conceptual y se propuso como solución y base de desarrollo de software el diseño. Con la creación de metodologías de diseño se dio pie a la creación de reglas y principios que permitieran garantizar los resultados esperados en el desarrollo del software. El conjunto de esta teoría permite el nacimiento de una nueva rama de la ingeniería denominada ingeniería del software.

La ingeniería de software es la aplicación de un método sistemático de creación, desarrollo y mantenimiento del software, en una definición más formal podemos decir “La ingeniería de software es la rama de la ingeniería que aplica los principios de las ciencias de la computación y las matemáticas para lograr resultados eficaces en los desarrollos de software”.

Algunas peculiaridades de la ingeniería de software son las siguientes:

- El producto de software es enteramente conceptual.
- No tiene propiedades físicas como peso, color o voltaje, por lo tanto no esta sujeto a leyes físicas.

- Su naturaleza conceptual crea un distanciamiento entre el software y el problema que el software resuelve. Difícil para una persona que entiende el problema entender el software que lo resuelve.
- Para probar es necesario contar con un sistema físico (Hardware).

El proceso de ingeniería de software es aquel en el que las necesidades de usuario son traducidas a requerimientos de software, estos requerimientos transformados a diseño y el diseño implementado en código, el código es probado y documentado para su uso operativo.

2.2.1. Ciclo de vida del software.

Todo proyecto de ingeniería tiene unos fines ligados a la obtención de un producto, proceso o servicio que es necesario generar a través de diversas actividades. Algunas de estas actividades pueden agruparse en fases porque globalmente contribuyen a obtener un producto intermedio, necesario para continuar hacia el producto final y facilitar la gestión del proyecto. Al conjunto de las fases empleadas se le denomina “ciclo de vida”.

El proceso de desarrollo de software requiere un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le conoce ciclo de vida del software y comprende tres grandes fases: concepción, construcción y mantenimiento.

2.2.2. Concepción.

Esta etapa nos permite recolectar todos los requisitos que debe cumplir el producto final de software. Esto permite establecer el alcance del proyecto y nos permite modelar la lógica del negocio que regirá a nuestro software así como definir la viabilidad del proyecto.

En esta etapa también se definen las interfaces que han de establecerse, tanto de usuario como interfaces con otros sistemas, la fuente de la información que ha de procesarse y los criterios de validación que se usaran para determinar el correcto funcionamiento del sistema. La etapa de concepción puede ser resumida en las siguientes actividades:

A. Análisis.

Constituye la recolección de requerimientos que permitirá desarrollar un modelo de la lógica a seguir. Se asegura que los requisitos son alcanzables

B. Diseño.

A partir del modelo de análisis se deducen las estructuras de datos, la estructura en la que descompone el sistema y la interfaz de usuario. Se identifican las soluciones tecnológicas para cada una de las funciones del sistema. Se definen métodos que validen el diseño.

C. Planificación.

Una vez establecido el modelo, se define una estrategia a seguir, se asignan recursos, se estiman costos, tiempos, se delegan tareas a los distintos integrantes del proyecto y se establece un plan de trabajo.

2.2.3. Construcción.

La fase de desarrollo consiste en la construcción del sistema. La construcción se realiza mediante la codificación, a un lenguaje de programación, del modelo provisto en la fase de concepción. En esta etapa se implementan las interfaces (de usuario o de otros sistemas) que servirán como entrada o salida de información del sistema. En esta etapa es posible distinguir las siguientes actividades:

A. Análisis y diseño de software.

En la parte de análisis podemos identificar requisitos, estructuras de los datos y algoritmos que existirán en el software. La parte de diseño nos define las características de interfaces y arquitecturas usadas.

B. Implementación

Es el proceso de codificación de la lógica del negocio traducido a un lenguaje de programación. El resultado de esta etapa es un programa ejecutable por la computadora que satisface las necesidades del usuario.

C. Pruebas

Esta etapa tiene la finalidad de asegurar que el funcionamiento del sistema es el pretendido. Se verifica que se cumplen con la calidad y requerimientos. En esta etapa también se prueba, si es el caso, el funcionamiento del sistema con otros sistemas involucrados.

2.2.4. Mantenimiento.

En esta fase, que tiene lugar después de la liberación del sistema, se asegura que el sistema siga funcionando y adaptándose a los nuevos requisitos.

2.2.5. Características de un buen software.

Podemos citar las siguientes características las siguientes características que debe cumplir un software hoy en día.

- Corrección: Se refiere a la facilidad que provee un software al momento de realizar un cambio motivado por un error en el modelo.
- Fiabilidad: Que los resultados obtenidos del sistema sean seguros, creíbles y sin error.

- Eficiencia: Se refiera a que se cumplan los objetivos o se obtengan los resultados óptimos mediante el uso mínimo de recursos.
- Integridad: Se refiere al buen funcionamiento entre la interfaz, la lógica del negocio y el despliegue de resultados.
- Facilidad de uso: El software debe resolver un problema al usuario no implicar otro.
- Facilidad de mantenimiento: Se refiere a la facilidad de cambios o funciones adicionales que pueda sufrir un sistema con el paso del tiempo.
- Flexibilidad: Susceptible de cambios o variaciones según las circunstancias o necesidades.
- Facilidad de prueba: Se refiere a la facilidad que existe para probar las diferentes funcionalidades del sistema.
- Portabilidad: Se refiere a la capacidad que tiene un sistema para ser ejecutado en distintos equipos no importando la arquitectura o sistema operativo de este.
- Facilidad de rehúso: Se refiere a la capacidad de las que tienen las distintas piezas de un sistema para ser utilizadas por otro.
- Interoperabilidad: Significa la capacidad del sistema para funcionar junto con otros sistemas.

2.2.6. Paradigmas de Programación.

En programación existen diferentes tipos de modelos básicos o filosofías que nos sirven como punto de partida para el diseño de soluciones informáticas, llamadas paradigmas. De esta manera el desarrollo de un sistema se realiza a partir de un modelo abstracto de programación.

Un paradigma de programación es una colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan, al final la estructura de un programa.

Esa estructura conceptual de modelos está pensada de forma que esos modelos determinan la forma correcta de los programas y controlan el modo en que pensamos y formulamos soluciones, y al llegar a la solución, ésta se debe de expresar mediante un lenguaje de programación. Para que este proceso sea efectivo las características del lenguaje deben reflejar adecuadamente los modelos conceptuales de ese paradigma.

Los paradigmas de programación no presentan una mejoría de uno con respecto a otro, sino que cada uno tiene sus ventajas y desventajas. También en la mayoría de las situaciones un paradigma resulta más apropiado que otro.

Paradigma Imperativo.

La programación en el paradigma imperativo consiste en determinar qué datos son requeridos para el cálculo, asociar a esos datos unas direcciones de memoria, y efectuar paso a paso una

secuencia de transformaciones en los datos almacenados, de forma que el estado final represente el resultado correcto.

En su forma pura este paradigma sólo soporta sentencias simples que modifican la memoria y efectúan bifurcaciones condicionales e incondicionales. Incluso cuando se añade una forma simple de abstracción procedimental, el modelo permanece básicamente sin cambiar. Los parámetros de los procedimientos son "alias" de las zonas de memoria, por lo que pueden alterar su valor, y no retorna ningún tipo de cálculo. La memoria también se puede actualizar directamente mediante referencias globales. Ejemplos de lenguajes en el paradigma estructurado son: C y Basic.

Características:

- Concepto de celda de memoria ("variable") para almacenar valores. El componente principal de la arquitectura es la memoria, compuesto por un gran número de celdas donde se almacenan los datos.
- Operaciones de asignación. Estrechamente ligado a la arquitectura de la memoria, se encuentra la idea de que cada valor calculado debe ser "almacenado", es decir asignado a una celda. Esta es la razón de la importancia de la sentencia de asignación en el paradigma imperativo.
- Repetición. Un programa imperativo, normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales, ya que en este modelo computacional la única forma de ejecutar algo complejo es repitiendo una secuencia de instrucciones.

Paradigma Funcional

El paradigma funcional está basado en el modelo matemático de composición funcional. En este modelo, el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado.

No existe el concepto de celda de memoria que es asignada o modificada. Más bien, existen valores intermedios que son el resultado de cálculos anteriores y las entradas a cálculos subsiguientes. Tampoco existen sentencias imperativas y todas las funciones tienen transparencia referencial. Ejemplos de estos lenguajes encontramos a LISP y Scheme.

Características:

- Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones
- Las funciones se construyen de manera puramente matemática.
- No existe la asignación de variables
- La iteración se consigue con la llamada a funciones recursivas.

Paradigma Heurístico.

Se puede definir como aquel tipo de programación computacional que aplica para la resolución de problemas reglas de buena lógica, denominadas heurísticas, las cuales proporcionan entre varios cursos de acción uno que presenta visos de ser el más prometedor, pero no garantiza necesariamente el curso de acción más efectivo.

Las problemas más comunes en los que se aplica este tipo de paradigma son problemas relacionados al tipo: "¿Cuál es el camino más corto?", "Listar todos los casos posibles", "¿Existe una disposición de elementos que satisfaga?, y demostraciones en general. Como ejemplos de lenguajes encontramos a Prolog.

Características:

- La programación se realiza en base a sentencias lógicas.
- Permite establecer reglas y restricciones sobre un conjunto de datos.
- Se permite la formulación de preguntas lógicas y con ello plantear hipótesis.

Paradigma Orientado a Objetos.

El paradigma de la programación orientada a objetos se basa en el concepto de objeto. El universo computacional está poblado por objetos, cada uno responsabilizándose por sí mismo, y comunicándose con los demás por medio de mensajes. Un objeto es aquello que tiene estado (propiedades), comportamiento (acciones y reacciones a mensajes) e identidad (propiedad que lo distingue de los demás objetos).

El enfoque orientado a objetos proporciona la capacidad para modelar e implementar soluciones en torno a los elementos del espacio del problema, es decir, la solución debe ser descrita en términos de las palabras que expresan el problema, en vez de en términos del sistema en el que se ejecute la solución final. Ejemplos de lenguajes de programación en este paradigma son Smalltalk, Java y C++.

Características.

- La unidad fundamental es el objeto, tratado como un componente que modela un elemento del problema.
- Los objetos se agrupan en clases que son abstracciones de atributos (físicos y lógicos) y características (funciones) que puede realizar un objeto.
- Los objetos implementan una parte pública, visible al resto de los objetos, y una parte privada modificable solo por el objeto mismo.

- La programación se realiza de forma modular, lo que permite la ejecución de componentes de software de manera independiente aumentando la posibilidad de rehusar código y reduciendo los costos de tiempo en cambios y mantenimiento.

2.3.Programación Orientada a Objetos.

El término programación orientada a objetos (POO) es un concepto desarrollado de técnicas de programación desde principios de la década de los setenta, aunque fue en la década de los noventa cuando aumento su difusión, uso y popularidad. No obstante se puede definir a la POO como una técnica o estilo de programación que utiliza objetos como bloque esencial de construcción.

La estructura y comportamiento de objetos similares están definidos en su clase común. Una clase es la definición de un conjunto de objetos que comparten una estructura y comportamiento. La definición de clases de objetos se logra gracias a la abstracción, es decir a la extracción de características principales que permiten diferenciar un objeto de otro.

Para entender mejor el concepto de abstracción, elemental en la programación orientada a objetos, podemos citar el siguiente ejemplo:

Consideremos 5 marcadores como los que se muestran en la figura 1.1, de este conjunto de objetos podemos decir que pertenecen a una misma clase de objetos llamada “marcador” y cada objeto tiene las características necesarias que lo identifican como un objeto “marcador” y no como un objeto de otro tipo como un disco compacto. Por medio de la abstracción podemos crear una clase que defina las características y comportamiento de cualquier objeto “marcador”.

Figura 1.1 Ejemplo de Objetos.

Podemos identificar las siguientes características y comportamientos de un “marcador”, descritas en la tabla 1.

Características	Comportamientos
Color	Pintar
Tamaño	Quitar tapa
Marca	Poner tapa
Nivel de tinta	Agregar tinta
Material	

Tabla 1.1 Características y comportamientos

Estas características y comportamientos son validos para todos los objetos de tipo “Marcador” que podamos citar. Las características de un conjunto de objetos definidos en una clase se le conoce como atributos y al conjunto de comportamientos reciben el nombre de métodos.

2.3.1. Abstracción

Todos los lenguajes de programación proporcionan abstracciones, podemos incluso decir que el lenguaje ensamblador es una pequeña abstracción de la maquina subyacente. Muchos de los lenguajes nominados “imperativos” (Fortran, BASIC y C) eran abstracciones a su vez del lenguaje ensamblador. Sin embargo estos lenguajes exigían pensar en términos de la estructura del computador más que en la del problema a resolver.

El proceso de abstracción puede ser visto como el proceso de definir las características más importantes de un objeto con la finalidad de crear nuevos tipos de datos que permitan modelar la solución en términos del problema mismo.

Los tipos abstractos de datos funcionan casi como los tipos de datos propios del lenguaje: es posible la creación de variables de un tipo (que se denominan objetos o instancias en el dialecto propio de la orientación a objetos) y manipular estas variables. Dado que una clase describe a un conjunto de objetos con características (datos) y comportamientos (funcionalidad) idénticos, una clase es realmente un tipo de dato, porque un número de punto flotante, también tiene un conjunto de características y comportamiento. La diferencia radica en que el programador define una clase para que encaje dentro del problema en vez de verse forzado a utilizar un tipo de dato existente que fue diseñado para representar una unidad de almacenamiento en una máquina.

2.3.2. Encapsulamiento

Una vez que se establece una clase, pueden construirse tantos objetos de esa clase como se desee y manipularlos como si fueran elementos que existen en el problema que se trata de resolver. Para conseguir que un objeto haga un trabajo útil dentro de un sistema, se debe realizar una petición como puede ser: ejecutar una transacción, dibujar algo en pantalla, realizar un cálculo, etc. Además cada objeto puede satisfacer determinadas peticiones. Las peticiones que se pueden hacer a un objeto se encuentran definidas en su interfaz. La representación gráfica de una clase como componente de software se muestra en la figura 1.2.

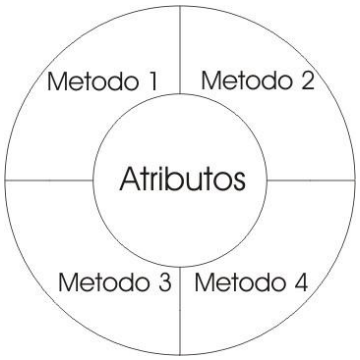


Figura 1.2. Representación gráfica de un componente de software

Esta representación muestra que para acceder a conocer o modificar el valor de un atributo, es necesario realizar la petición al método correspondiente, es decir, la interfaz establece que peticiones pueden hacerse a un objeto particular. Sin embargo debe haber código en algún lugar que constituya esas peticiones. Éste junto con los datos ocultos constituye el encapsulamiento.

El encapsulamiento permite reducir el potencial de errores que pudieran ocurrir. En un sistema que consta de objetos, éstos dependen unos de otros en diversas formas. Si uno de ellos falla y los especialistas de software tienen que modificarlo de alguna forma, el ocultar sus operaciones de otros objetos significaría que tal vez no será necesario modificar los demás objetos. Esto permite realizar programas modulares, lo que significa un tiempo mínimo en cuestiones de mantenimiento, que se traducen a realizar un cambio de la pieza de software afectada dentro del sistema.

Para garantizar que un objeto de software realice su trabajo correctamente, cada clase define una parte de su interfaz como pública y otra como privada. La parte pública puede ser accedida y modificada libremente por otra pieza de software y la parte privada queda reservada para el manejo del objeto mismo.

El diseño de clases basadas en el encapsulamiento se basa en la idea del diseño ideal de clases, en la práctica se suele permitir la modificación de ciertos atributos de manera directa por cuestiones de eficiencia.

2.3.3. Herencia.

Una vez que se ha definido y probado una clase, idealmente representa una unidad útil de código. Si se logra un buen diseño, puede que se desee reutilizar dichas piezas de software. La reutilización de código es una de las mayores ventajas que proporcionan los lenguajes de programación orientados a objetos.

La manera más sencilla de reutilizar una clase es simplemente usar un objeto de esa clase directamente, pero también es posible ubicar un objeto de esa clase dentro de otra clase. La

nueva clase puede construirse a partir de un número indefinido de otros objetos, de igual o distinto tipo. Dado que se está construyendo una nueva clase a partir de otras existentes, a este concepto se le denomina composición. La composición se suele representar mediante una relación "es parte de", como por ejemplo, "el motor es parte de un coche", este concepto se representa gráficamente como muestra en la figura 1.3.

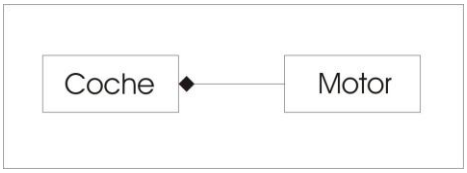


Figura 1.3 Ejemplo de composición.

La composición conlleva una gran carga de flexibilidad. De esta manera es posible cambiar los objetos de una clase en tiempo de ejecución, para así cambiar de manera dinámica el comportamiento de un programa.

Existen ocasiones en que creamos una clase y posteriormente deseamos crear otra clase que podría tener una funcionalidad similar. Sería mejor si pudiéramos hacer uso de una clase ya existente, clonarla, y después hacer al "clon" las adiciones y modificaciones que sean necesarias. Este proceso se logra mediante la herencia, con la excepción de que si se cambia la clase original (denominada clase base, súper clase o clase padre) el "clon" modificado (denominado clase derivada, clase heredada, sub clase o clase hija) también reflejaría esos cambios.

La herencia expresa esta semejanza entre tipos haciendo uso del concepto de tipos base y tipos derivados. Un tipo base contiene todas las características y comportamientos que comparten los tipos que de él se derivan. A partir del tipo base, es posible derivar otros tipos para expresar las distintas maneras de llevar a cabo esta idea.

Por ejemplo, consideremos el uso de las figuras geométricas en un sistema de diseño. El tipo base es "figura" y cada una de ellas tiene un tamaño, color, posición, etc. Cada figura puede moverse, dibujarse, borrarse, colorearse, etc. A partir de esta se pueden derivar (heredar) figuras específicas: círculos, cuadrados, triángulos, etc., pudiendo tener cada uno características y comportamientos adicionales. Algunos comportamientos pueden ser distintos, como pudiera ser el cálculo del área de los distintos tipos de figura.

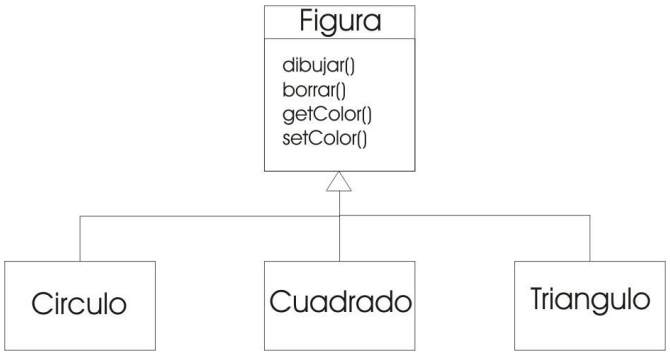


Figura 1.4 Representación gráfica de la herencia

Con los objetos, el modelo principal lo constituye la jerarquía de tipos, de manera que se puede ir directamente de la descripción del sistema en el mundo real a la descripción del sistema en código.

Dado que sabemos el tipo de una clase en base a los mensajes que se le pueden enviar, la clase derivada es del mismo tipo que la clase base. Así en el ejemplo anterior, “un círculo es una figura”.

2.3.4. Polimorfismo

Al tratar con las jerarquías de tipos, a menudo se desea tratar un objeto no como el tipo especifico que es, sino como su tipo base. Esto permite escribir código que no depende de tipos específicos. En el ejemplo de los figuras, las funciones manipulan figuras genéricas sin que importe si son círculos, cuadrados, triángulos o cualquier otro tipo de polígono que no se ha definido aun. Todas las figuras pueden dibujarse, borrarse y moverse, por lo que estas funciones simplemente envían un mensaje a un objeto figura.

Cuando hacemos referencia de tipo figura a un objeto circulo, la misma referencia figura puede ser utilizada para hacer referencia a un objeto de tipo cuadrado, puesto que un circulo y un cuadrado comparten la clase base “Figura”. Esto quiere decir que una referencia de tipo figura puede adoptar varias formas, un circulo, un cuadrado o un triangulo, esta capacidad de adoptar varias formas se le conoce como polimorfismo.

El proceso de tratar un tipo derivado como si fuera el tipo base se le denomina conversión de tipos (moldeado) hacia arriba. El nombre moldear (cast) se utiliza en el sentido de moldear (convertir) un molde, y es hacia arriba siguiendo la manera en que se representa el diagrama, con los tipos base en la parte superior y las derivadas colgando hacia abajo.

Un programa orientado a objetos siempre tiene algún moldeado hacia arriba pues ésta es la manera de desvincularse de tener que conocer el tipo exacto con que se trabaja en cada instante.

2.4.Patrones de Diseño.

Los patrones de diseño son maneras convenientes de solucionar un problema común de programación, la idea detrás de los patrones de diseño es simple, escribir las estructuras de programación e interacción entre objetos que los programadores han encontrado útiles para resolver problemas frecuentes.

El campo de patrones de diseño nace años atrás cerca de 1980. En ese tiempo predominaba la programación estructurada y la programación orientada a objetos no era ampliamente apoyada, siendo SmallTalk el lenguaje más común orientado a objetos. La idea de programar marcos de trabajo o frameworks fue muy popular, algunos de esos frameworks desarrollados hoy los denominamos patrones de diseño.

Los patrones de diseño se clasifican regularmente en 3 tipos de acuerdo al problema que resuelven:

- Patrones creacionales.
- Patrones estructurales.
- Patrones de comportamiento.

2.4.1. Patrones Creacionales.

Los patrones de creación tratan de asegurar la mejor manera de crear objetos o instancias de clase. Esto es importante ya que un programa no debería depender de cómo los objetos son creados, sin embargo, esto puede causar que se eleve el nivel de complejidad. En muchos casos, la forma en que se crea el objeto puede variar en función de las necesidades del programa y un proceso especial de “creación” de objetos puede hacer que un programa sea más flexible y general.

Algunos de los patrones de creación más comunes son:

- El método de fábrica (The Factory Method): Proporciona una toma de decisión de clase que retorna un tipo de objeto de varias posibles subclases de una clase abstracta base dependiendo del dato que le proporcionemos.
- Fabrica Abstracta (Abstract Factory): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- Constructor Virtual (Builder): Abstrae la construcción de un objeto complejo para su representación, de esta forma varias de las diferentes presentaciones pueden ser creadas dependiendo de las necesidades del programa.
- Prototipo (Prototype): Crea nuevos objetos clonándolos de una instancia ya existente.

- Instancia única (Singleton): Es una clase en la cual no se puede crear más de una instancia. También proporciona un simple punto de acceso a la instancia.

2.4.2. Patrones Estructurales.

Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras. La diferencia entre patrones de clase y patrones de objetos es que los patrones de clase describen como la herencia puede ser usada para proporcionar más interfaces de programa más útiles. Los patrones de objeto, por otra parte, describen como los objetos pueden estar compuestos en grandes estructuras usando composición o incluso objetos dentro de otros objetos.

Los patrones estructurales más utilizados se listan a continuación:

- Adaptador (Adapter): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- Peso Ligero (Flyweight): Es utilizado para compartir objetos, donde cada instancia no contiene su propio estado, sino que se almacena en el exterior. Esto permite compartir objetos eficientemente y salva espacio, cuando hay demasiadas instancias pero solo pocas con diferente estados.
- Fachada (Facade): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- Puente (Bridge): Desacopla la interfaz de un objeto de su aplicación.
- Envoltorio (Decorator): Añade funcionalidad a un objeto dinámicamente.

2.4.3. Patrones de Comportamiento.

Los patrones de comportamiento son los patrones especializados en realizar la comunicación entre objetos.

Los patrones de comportamiento más comunes son:

- Observador (Observer): define una dependencia de un número de clases que son notificadas de un cambio.
- Mediador (Mediator): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- Cadena de responsabilidad (Chain of Responsibility): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- Visitante (The visitor): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.
- Orden (Command): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- Modelo Vista Controlador (MVC): El patrón de modelo vista controlador es una consecuencia del patrón observador aplicado a un conjunto de clases que realizan

diferentes tareas individuales para un resultado en conjunto. En el modelo vista controlador los datos son representados por un Modelo, la vista por un componente visual y el controlador es la comunicación entre el modelo y la vista.

2.5.UML

El Lenguaje de Modelado Unificado (UML) es un lenguaje para especificar, visualizar, construir y documentar componentes de sistemas de software, así como modelar procesos de negocio y otros sistemas no software. El UML presenta un conjunto de las mejores prácticas que han dado éxito en el modelado de grandes y complejos sistemas.

2.5.1. Metas de UML

Las principales metas en el diseño con UML son las siguientes:

1. Proporcionar a los usuarios un modelo expresivo y visual listo para usarse con el que pueden desarrollar e intercambiar ideas. El UML consolida un conjunto de conceptos básicos generalmente aceptados por varios métodos y herramientas de modelado.
2. Permitir la expansión y especialización de mecanismos para detallar los conceptos básicos. El UML se adapta a las nuevas necesidades de temas específicos. Por lo tanto se cree que los mecanismos de expansión deben soportar variantes a los casos comunes. Los conceptos básicos no deben ser cambiados a menos que sea necesario. El uso del UML, permitirá:
    - Construir modelos usando los conceptos básicos sin usar los mecanismos de expansión para la mayoría de las aplicaciones.
    - Añadir los nuevos conceptos y notaciones para los problemas no cubiertos por las bases del UML.
    - Elegir entre las variantes de conceptos existentes cuando no exista un consenso claro.
    - Especializar los conceptos, notaciones y restricciones de las aplicaciones en particular.
3. Ser independiente de los lenguajes de programación y de los procesos de desarrollo. El UML soporta todos los lenguajes de programación. Además puede soportar varios métodos y procesos de generación de modelos. El UML puede soportar múltiples lenguajes de programación y desarrollar métodos sin mucha dificultad.
4. Proporcionar una base formal para la comprensión del lenguaje de modelado. Debido a que se necesita de la formalidad para ayudar a entender el lenguaje, el UML debe ser preciso y comprensible. El UML permite una definición formal del formato estático del modelo usando un modelo expresado en los diagramas de clases. Este es un enfoque formal popular y ampliamente aceptado para especificar el formato de un modelo que

directamente lleva al intercambio de ideas. El UML expresa el manejo operacional de la mayoría de los desarrollos en un lenguaje natural y preciso.

5. Fomentar el crecimiento del mercado de herramientas orientadas a objetos. Cuando se utiliza un lenguaje de modelado estándar usado por la mayoría de desarrolladores y herramientas, se beneficia directamente el mercado de herramientas orientadas a objetos.
6. Soportar un alto nivel de conceptos de desarrollo tales como las colaboraciones, los frameworks, los patrones y los componentes. Es esencial definir la semántica de los conceptos lograr todos los beneficios de la orientación a objetos y su reutilización.
7. Integrar las mejores prácticas. La principal motivación detrás del desarrollo del UML ha sido integrar las mejores prácticas de la industria, que abarquen ampliamente las diferentes vistas basadas en niveles de abstracción, dominios, arquitecturas, etapas de ciclos de vida, implementación de tecnologías, etc. Por lo tanto, el UML es la integración de las mejores prácticas.

2.5.2. Uso de UML

Como el valor estratégico de software se incrementa para muchas empresas, la industria busca la manera de automatizar la producción de software y así entregar calidad y reducir el costo y tiempo de desarrollo. Estas técnicas incluyen componentes tecnológicos, programación visual, patrones y frameworks. Los negocios también buscan técnicas para administrar la complejidad de los sistemas cuando aumentan de alcance. En particular, es necesario resolver problemas de arquitectura, problemas de distribución, concurrencia, replicación, seguridad, balance de carga de trabajo y tolerancia a fallos. El UML fue diseñado para responder a estas necesidades.

2.5.3. Diagramas que componen el UML

El UML tiene muchas formas gráficas que al combinarse forman diagramas. La finalidad de estos diagramas es ofrecer varias perspectivas del sistema que en su conjunto forman un modelo. Cabe mencionar que los diagramas UML describen lo que debe hacer el sistema, sin embargo no ahondan en cómo implementarlo. A continuación se describen los diagramas más comunes del UML, sin embargo, estos pueden combinarse para construir diagramas híbridos, lo cual ayuda a organizar el modelo de distintas maneras, y así dar la posibilidad de extender su significado.

2.5.4. Diagramas de Clase

Un diagrama de clases muestra los atributos (propiedades) y acciones que puede realizar las clases. Por ejemplo, si se piensa en una clase Auto, el cual tiene como atributos: marca, color, número de serie, kilometraje y capacidad de pasajeros. Donde algunas de las acciones que puede realizar son: avanzar, retroceder, dar vuelta y detener. Entonces la figura 1.5 muestra la simbología para representar a la clase Auto.

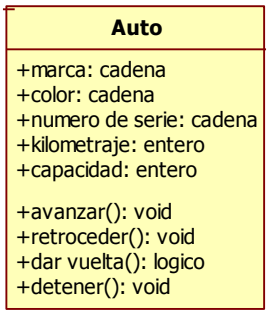


Figura 1.5 clase Auto

El objetivo de pensar de esta manera, es que en la mayoría del software moderno se pretende simular algún aspecto del mundo real, con ello es mas sencillo desarrollar aplicaciones además de que esta clase de diagramas ayudan a los programadores a trabajar de una manera más sencilla y hablarle a los usuarios en su mismo idioma, lo que hace más fácil la explicación de aquellos detalles sobre los problemas que se pretenden resolver con el sistema.

2.5.5. Diagramas de Objetos

Un objeto es una instancia de una clase, es decir, una entidad con valores específicos. Por ejemplo, si de la clase Auto, se tiene el siguiente objeto: marca Acura, color negro, número de serie DJAS47274S, kilometraje 12 239 Km., capacidad de 5 personas. Entonces el elemento gráfico para representarlo es un rectángulo con el texto subrayado donde antes de los dos puntos (:) se pone el nombre del objeto y después la clase a la que pertenece.

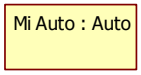


Figura 1.6 Diagrama de objeto

2.5.6. Diagrama de casos de uso

Un caso de uso es una historia sobre las acciones que hace un sistema desde el punto de vista del usuario. Es una herramienta valiosa para la obtención de requerimientos del usuario. La forma de representar a los casos de uso se muestra en la figura 1.7.

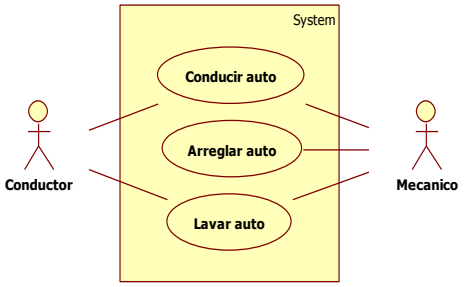


Figura 1.7 Caso de uso

La figura que representa al Conductor o al Mecánico, se conoce como Actor y es quien inicia el caso de uso en el sistema. Cabe mencionar que en los diagramas de casos de uso existen dos elementos gráficos importantes con los siguientes propósitos: incluir casos de uso (modularización) y extender los casos de uso para que se puedan dar variaciones o particularidades a partir de otros casos de uso (herencia).

Inclusión. Para el caso de uso conducir, se pueden incluir los casos de uso: avanzar, retroceder, dar vuelta y detener. La figura 1.8 lo representa:

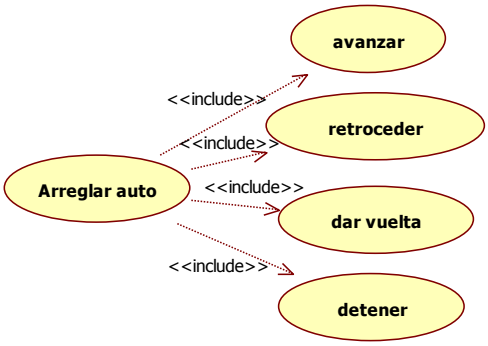


Figura 1.8 Inclusión de casos de uso

Extensión. Para el caso de uso arreglar auto, pueden existir variaciones o particularidades como se muestra en el ejemplo:

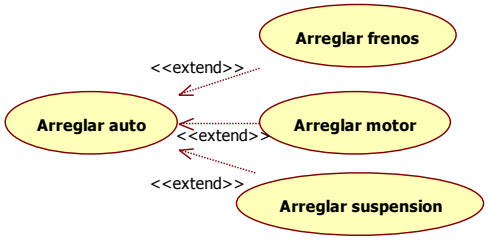


Figura 1.9 Extensión de casos de uso.

2.5.7. Diagrama de Estados.

Un estado es la condición de un objeto, cuando esta condición se cumple, se ejecuta una acción o se espera por un evento. Un estado se define por los valores de los atributos del objeto o por la existencia de enlace con otro objeto. Un diagrama de estados y transiciones muestra todos los mensajes que un objeto puede enviar o recibir, presentando los estados en los que se puede encontrar un objeto. Un ejemplo se muestra en la figura 1.10.

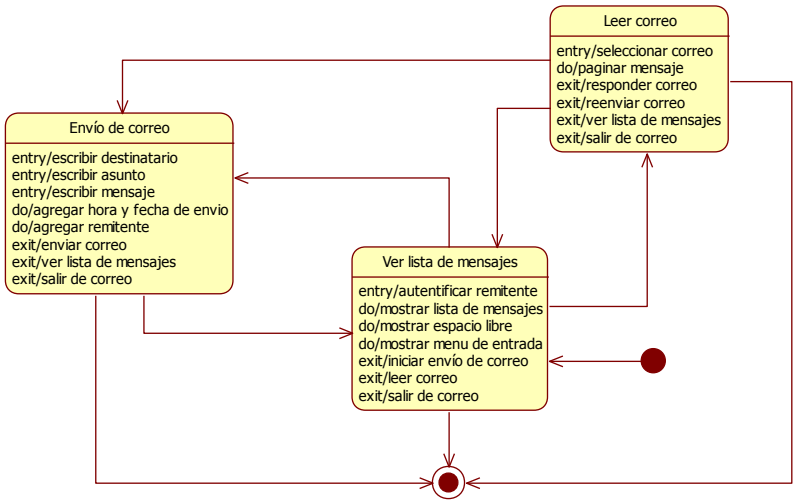


Figura 1.10 Diagrama de Estados

Este diagrama representa el envío de correos electrónicos a través de un WEBMAIL. Como se puede notar, existen tres momentos en un estado para realizar acciones: al inicio (entry), durante (do) y al salir (exit).

2.5.8. Diagrama de Secuencias.

Un diagrama de secuencias se compone por objetos representados con rectángulos con el nombre subrayado, mensajes representados con líneas continuas con punta de flecha y por el tiempo representado por un rectángulo a través de una línea vertical punteada. El siguiente ejemplo de la figura 1.11, muestra un diagrama de secuencias para el alta de un paciente hecha por un terapeuta en el sistema de administración de la clínica donde trabaja:

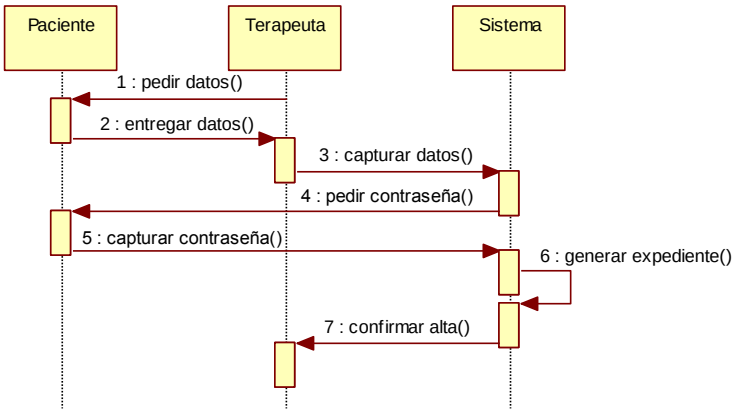


Figura 1.11 Diagrama de secuencias.

2.5.9. Diagrama de Colaboraciones

El diagrama de colaboraciones, muestra a los objetos interactuando a partir de mensajes entre sí. Para representar un mensaje se dibuja una flecha entre cada objeto. Cabe señalar que todo diagrama de secuencias tiene su equivalente en el diagrama de colaboraciones. Por consiguiente, la figura 1.12 muestra el equivalente al diagrama de secuencias donde un terapeuta da de alta a un paciente en el sistema de la clínica donde trabaja.

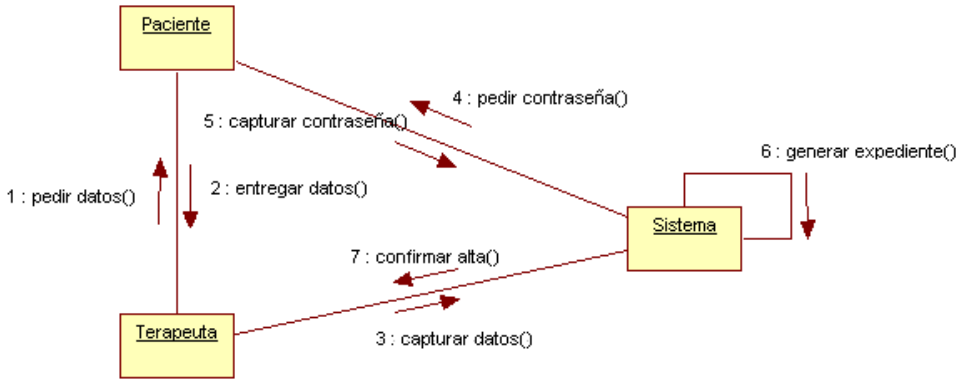


Figura 1.12 Diagrama de colaboraciones.

La tabla 1.2 muestra lo que se puede representar en un diagrama de colaboraciones:

Evento	Representación Gráfica
Acción	1: retorno := acción(param1, param2, ..., paramN)
Envío de valor	1: valor
Recepción de valor	1: valor
Creación de objeto	<<create>> 1: retorno := acción(param1, param2, ..., paramN)
Destrucción de objeto	<<destroy>> 1: retorno := acción(param1, param2, ..., paramN)
Objeto	Objeto

Tabla 1.2 elementos del diagrama de colaboraciones

2.5.10. Diagrama de Actividades

El diagrama de actividades muestra lo que ocurre durante la ejecución de un proceso. Este diagrama es similar a los diagramas de flujo, ya que muestra las actividades a realizar, existen bifurcaciones, hay decisiones y se muestra la secuencia de pasos a seguir. Los componentes gráficos del diagrama de actividades se muestran en la tabla 1.3.

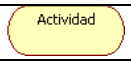
Nombre	Representación Gráfica	Uso
Estado inicial		Se usa como punto inicial del diagrama de actividades, solo puede existir una vez en el diagrama y se asocia al objeto con el que inicia el proceso.
Estado Final		Muestra el punto final del diagrama de actividades, este se puede repetir varias veces para mejorar el entendimiento del diagrama.
Carril		Denota la frontera de responsabilidades de un objeto
Transición		Indica la transición entre una actividad y otra
Decisión		Se usa para definir un punto de decisión entre una actividad y otra(s).
Actividad		Describe la actividad que realiza el objeto

Tabla 1.3 Elementos del diagrama de actividades.

El siguiente diagrama de actividades, de la figura 1.13, muestra la forma en que un paciente interactúa con el sistema de calificación de pruebas psicológicas en un centro de ayuda contra adicciones.

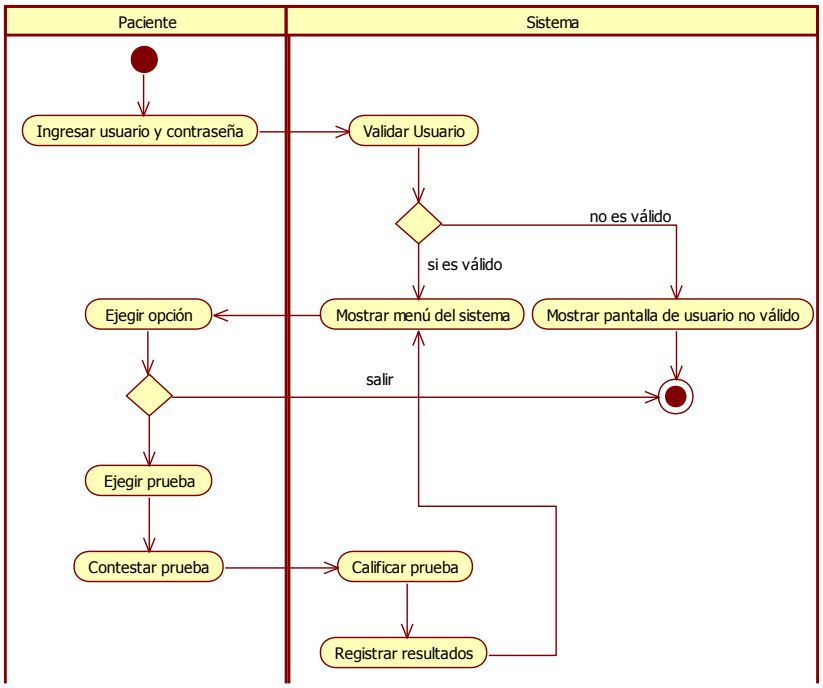


Figura 1.13 Diagrama de actividades

2.6.Bases de Datos

Una base de datos es un almacén que nos permite guardar grandes cantidades de información de manera organizada en memoria auxiliar. Esta funciona como núcleo de los sistemas transaccionales, ya que permite manipular los datos de entrada y salida de manera eficiente con operaciones como mostrar, agregar, modificar o eliminar registros.

2.6.1. Características de las Bases de Datos

Las bases de datos presentan particularidades de funcionamiento que ofrecen ventajas en comparación al almacenamiento de información en sistemas de archivos. Enseguida se enlistan algunas de sus características más importantes.

Independencia lógica y física: Significa que la base de datos es un sistema independiente al software que funciona como interfaz entre los datos y el usuario física y lógicamente. Esta es una ventaja que facilita el mantenimiento y crecimiento de las aplicaciones ya que los datos están totalmente independientes de la funcionalidad de la aplicación.

Redundancia mínima: Se refiere a que la base de datos almacena la información en archivos estructurados y dependiendo del modelado se puede evitar por completo la información repetida. Sin embargo, existen casos en los que es necesario redundar para optimizar consultas frecuentes y así facilitar el acceso a las aplicaciones, es decir, se sacrifica espacio para mejorar el rendimiento.

Acceso concurrente por parte de múltiples usuarios: En la mayoría de los casos lo más común es que muchas personas necesiten acceder a una base de datos, ya sea para recuperar información o para almacenarla. Y es también frecuente que dichos accesos se realicen de forma simultánea. Es por eso que el manejador de la base de datos (DBMS) debe controlar el acceso concurrente a la información y así evitar las inconsistencias.

Integridad de datos: Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada.

Versatilidad para representar la información: Significa que una base de datos ofrece la ventaja de mostrar diferentes perspectivas o vistas de la información almacenada.

Seguridad: La información almacenada en una base de datos puede llegar a tener un gran valor. Y es necesario que los Manejadores de Bases de Datos (DBMS) garanticen que esta información se encuentra asegurada frente a usuarios malintencionados, que intenten leer información privilegiada o que desean manipular o destruir la información.

Respaldo y recuperación: Los DBMS deben proporcionar una forma eficiente de realizar copias de seguridad de la información almacenada en ellos, y de restaurarlas para evitar la pérdida de datos.

2.6.2. Diseño de Bases de Datos Relacionales.

El objetivo del diseño de una base de datos relacional es lograr que los datos sean almacenados con un mínimo de redundancia. Una de las formas de llegar al objetivo consiste en diseñar esquemas que tengan una forma normal adecuada. Para determinar si se tiene una de las formas normales, se requiere mayor información del mundo real que se intenta modelar en una base de datos. Igualmente el diseño relacional contempla una serie de limitaciones producto de las dependencias de datos.

2.6.3. Fases del diseño de una base de datos relacional.

El diseño de una base de datos relacional pasa por las siguientes etapas:

Necesidad de representar un fenómeno del mundo real: El primer paso es darse cuenta de que es necesario representar en un modelo de datos un proceso que ocurre en el mundo real, tal representación puede ser parcial o total y se define mediante las necesidades de información de los usuarios que van a consultar la base de datos por cualquiera de sus interfaces.

Recolección y Análisis de Requerimientos: Esta etapa se trata de reunir la información que los futuros usuarios nos pueden dar sobre su problemática a partir de entrevistas con la finalidad de recolectar, documentar y analizar para ofrecer una o varias alternativas de solución de acuerdo a sus necesidades.
Diseño Conceptual: En cuanto los requerimientos se han concretado a partir de ellos se procede a construir un modelo conceptual llamado también modelo de entidad relación, en este se describen los grupos de atributos en entidades y las relaciones que guardan entre ellos.

- Entidad: Es un objeto que representa a un grupo de información sobre un tema determinado, el nombre se construye a partir de un concepto abstracto, una cosa, una persona o un suceso. Por ejemplo: Factura, Empleado, Cliente, Ventas, Cuentas por Pagar, etc. La forma de representarlas es mediante un rectángulo donde su nombre aparece en el interior, en un modelo conceptual el nombre debe ser único respecto a las otras entidades que lo componen. Existen dos tipos de entidades: fuertes y débiles. Una entidad fuerte es aquella que no depende de la existencia de otra entidad, y una débil es la que sí depende. Ejemplo: Si en un modelo conceptual la entidad cliente y la entidad familiar representan que un cliente pude tener uno o más familiares registrados en el sistema, entonces la entidad cliente es una entidad fuerte y la entidad familiar es una entidad débil, ya que para que exista un familiar, debe estar registrado un cliente al que se le pueda asociar. La entidad débil se representa gráficamente con un doble rectángulo, mientras que entidad fuerte con un rectángulo simple.

- Relación: Es la forma de expresar la correspondencia entre dos entidades, su nombre se construye a partir de la función que las une y gráficamente se representa con un rombo con el nombre al interior. Por ejemplo: Si tenemos una entidad Proveedor y una Artículo, el nombre de la relación podría ser Vende, entonces se puede representar que un Proveedor Vende uno o más Artículos. El número de entidades que participan en una relación, determinan el grado de la misma, es decir si en una relación participan 2 entidades, se dice que el grado de la relación es 2 o binaria, si participan 3 es de grado 3 o ternaria y así sucesivamente. También existen relaciones llamadas recursivas, donde sólo aparece una entidad y se dice que se relaciona a sí misma. Por ejemplo: Si se tiene una entidad llamada Empleado, donde cada empleado tiene un jefe directo, y a su vez el jefe directo es también un empleado, entonces se tiene la relación recursiva que se podría llamar Le Reporta A, y quedaría como: Un Empleado le reporta a un empleado, siendo el segundo el jefe directo. Otro concepto relacionado con las relaciones es la cardinalidad, la cual define el grado de participación entre una entidad y otra con un número mínimo y uno máximo.
- Atributo: Es el elemento que representa una característica o un hecho de interés sobre una entidad o relación, es decir determinan las propiedades básicas en las entidades y relaciones. Los atributos se representan gráficamente por formas circulares interconectadas por medio de una línea con la entidad o relación correspondiente. Existen dos tipos de atributos, los simples y los compuestos, los simples se representan con círculos y son aquellos que son indivisibles por sí mismos, mientras que los atributos compuestos se representan por óvalos y se pueden dividir en más atributos compuestos o simples. Por ejemplo el atributo Dirección es un atributo compuesto, ya que se puede dividir en otros atributos como son: Calle, Número, Delegación o Municipio, Colonia, Código Postal, Ciudad, etc. Mientras que un atributo simple tomando el mismo ejemplo podría ser la Colonia. Un atributo también se le puede clasificar como monovaluado o multivaluado, un atributo monovaluado es aquel donde la relación entre el atributo y el valor es uno a uno mientras que en el multivaluado guarda una relación uno a muchos con los valores, por ejemplo un atributo Dueño de la entidad Auto es monovaluado ya que un auto solo puede pertenecer a un Dueño, mientras que el atributo Color es multivaluado, ya que un auto puede estar pintado de varios colores. Los atributos monovaluados se representan gráficamente con una línea simple, mientras que los multivaluados con doble línea. Otra clasificación de los atributos que se debe tomar en cuenta, es la que determina si estos son Almacenados o Derivados, un atributo almacenado es aquel que no es calculado mediante otros atributos, mientras que el derivado si y es representado gráficamente por una línea punteada. Por ejemplo, el atributo Fecha de Nacimiento en la entidad Alumno, es un atributo almacenado ya que no surgió de ningún cálculo, mientras que el atributo Edad, es derivado ya que se puede calcular mediante el anterior. Finalmente cabe mencionar que al conjunto de valores posibles para un atributo le denomina dominio.

- Identificador: Un identificador es uno o más atributos que hacen única a cada instancia en la entidad. Es importante que cada entidad tenga determinado un identificador. Las condiciones para comprobar si un identificador está correctamente definido son las siguientes:
    1. No pueden existir dos instancias con el mismo valor en su identificador.
    2. Si se omite cualquiera de los atributos que conformen el identificador, la condición anterior deja de cumplirse.

  Para representar a un identificador, basta subrayar a los atributos que lo componen.
- Elementos gráficos: Son todos los componentes visuales que nos ayudan a representar las reglas de negocio en un Modelo Conceptual , la figura 1.14 muestra sus elementos:

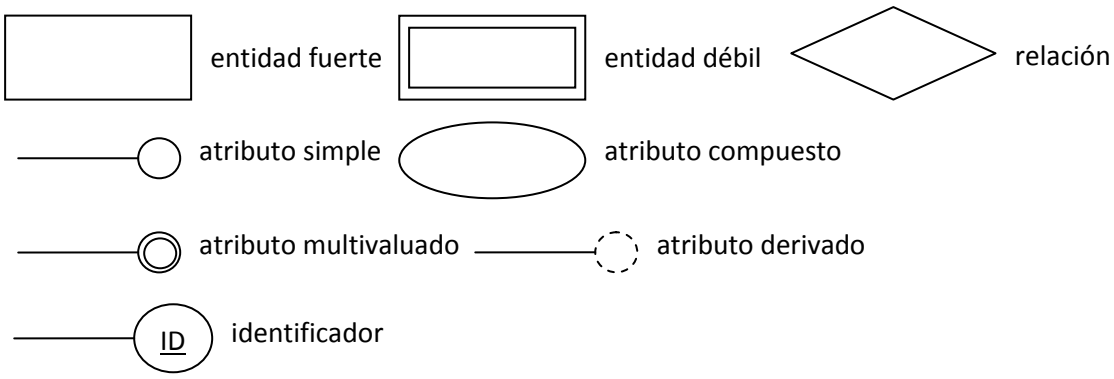


Figura 1.14 Elementos gráficos de la base de datos.

Diseño Lógico: El siguiente paso es desarrollar el modelo lógico, la forma de hacerlo es usando una herramienta de software de diseño de bases de datos, el objetivo es transformar el modelo conceptual a un modelo de datos relacional donde su estructura debe estar totalmente normalizada (como mínimo a la 3ª forma normal). Las formas normales se describen enseguida:

- 1ª forma normal: Una entidad está en primera forma normal si y solo si:
    1. Todos los atributos de la entidad son atómicos (indivisibles).
    2. La entidad tiene un identificador.
- 2ª forma normal: Una entidad está en segunda forma normal si y solo si:
    1. La entidad está en 1ª forma normal.
    2. Los atributos que no pertenecen al identificador compuesto dependen completamente este, es decir que no existen dependencias parciales.
- 3ª forma normal: Una entidad está en tercera forma normal si y solo si:
    1. La entidad está en 2ª forma normal.
    2. Cada atributo que no forma parte de ningún identificador, depende directamente y no transitivamente, del identificador.

En esta etapa se debe definir el diccionario de datos para cada entidad, donde los elementos que lo componen comúnmente son los siguientes: identificador, entidad, nombre del atributo,

descripción, dominio, tipo de dato, longitud y cómo se calcula en caso de atributos derivados. La tabla 1.5 muestra un ejemplo de diccionario de datos:

Id	Entidad	Nombre	Descripción	Dominio	Tipo de dato	Longitud	Decimales	Cálculo
1	Empleado	aplldo_ptrno	Apellido paterno del empleado	A-z y espacios	varchar	60	N/A	N/A
2	Empleado	aplldo_mtrno	Apellido materno del empleado	A-z y espacios	varchar	60	N/A	N/A
3	Empleado	nmbres	Nombres del empleado	A-z y espacios	varchar	60	N/A	N/A
4	Empleado	fcha_ncmnto	Fecha de nacimiento del empleado	Fecha en formato AAAA-MM-DD, donde AAAA es año, MM mes y DD día	date	N/A	N/A	N/A
5	Empleado	edad	Edad del empleado	Números >0	smallint	N/A	N/A	Tomando la fecha actual y la fecha de nacimiento, aplicar la resta y calcular los años

Tabla 1.5 Diccionario de datos.

Es preferible que el diccionario de datos se encuentre almacenado en la base de datos para conocimiento de los usuarios y del administrador de la base de datos.

Diseño físico: El diseño físico se realiza a partir del lógico, donde se define la estructura de almacenamiento interna y la organización de los archivos de la base de datos, sin embargo puede que por la naturaleza del negocio se deba “desnormalizar” la estructura para aumentar el rendimiento, este caso únicamente aplica para sistemas de consulta, mientras que para sistemas transaccionales, una estructura normalizada es la más conveniente para realizar las operaciones de entrada y salida de información hacia la base de datos. En esta etapa, se definen índices para agilizar búsquedas, constraints para delimitar el dominio de los datos o para asegurar que se cumplan las dependencias y asegurar la integridad y en ocasiones se definen triggers que son acciones que se aplican según condiciones después de que se suscita algún evento.

2.6.4. Reglas de Codd.

Edgar Frank Codd científico inglés (23 de agosto de 1923 - 18 de abril de 2003) conocido por sus aportes a la teoría de bases de datos relacionales, publicó 12 reglas que un verdadero sistema de bases de datos relacional debería tener, aunque en realidad cumplirlas por completo resulta muy complicado, por lo que estas reglas únicamente se usan para estimar que tan relacional es un sistema (entre más se cumplan, es más relacional). Enseguida se enuncian:

1. *Regla de la información*: "Toda la información, incluyendo nombres de entidades, nombres de vistas, nombres de atributos, y los datos de los atributos deben estar almacenados en tablas dentro de las bases de datos. Las tablas que contienen tal información constituyen el Diccionario de Datos. Esto significa que todo tiene que estar almacenado en las tablas."
2. *La regla del acceso garantizado*: "Dado un nombre de entidad, el valor del identificador, y dado el nombre del atributo requerido, deberá encontrarse uno y solamente un valor. Por esta razón la definición de identificadores para toda entidad es prácticamente obligatoria."
3. *Tratamiento sistemático de los valores nulos*: "La información inaplicable o faltante, debe ser representada con valores nulos y el manejador de bases de datos, debe ser capaz de soportarlos."
4. *La regla de descripción de la base de datos*: "La estructura de la base de datos debe ser almacenada en entidades y atributos para que sea accesible a usuarios autorizados."
5. *La regla del sublenguaje integral*: "Debe haber al menos un lenguaje que sea integral para soportar la definición de datos, manipulación de datos, definición de vistas, restricciones de integridad, y control de autorizaciones y transacciones. Esto significa que debe haber por lo menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos."
6. *Regla de la actualización de Vistas*: "Todas las vistas que asií lo requieran deben actualizarse automáticamente por el manejador de la base de datos."
7. *La regla de insertar y actualizar*: "Esto significa que las cláusulas SELECT, UPDATE, DELETE e INSERT deben estar disponibles y operables sobre los registros, independientemente del tipo de relaciones y restricciones que haya entre las entidades."
8. *La regla de independencia física*: "El comportamiento de los programas de aplicación y de la actividad de usuarios vía terminales debería ser predecible basados en la definición lógica de la base de datos, y éste comportamiento debería permanecer inalterado, independientemente de los cambios en la definición física de ésta."
9. *La regla de independencia lógica*: "La independencia lógica de los datos, especifica que los programas de aplicación y las actividades de terminal deben ser independientes de la estructura lógica, por lo tanto los cambios en la estructura lógica no deben alterar o modificar estos programas de aplicación."
10. *La regla de la independencia de la integridad*: "Todas las restricciones de integridad deben ser definibles en los datos, y almacenables en el catalogo, no en el programa de aplicación. Las reglas de integridad son: ningún componente del identificador puede tener valores en blanco o nulos. (esta es la norma básica de integridad) además de que para cada valor de la llave foránea deberá existir un valor de llave primaria concordante. La combinación de estas reglas aseguran que haya Integridad referencial."
11. *La regla de la distribución*: "El sistema debe poseer un lenguaje de datos que pueda soportar que la base de datos esté distribuida físicamente en distintos lugares sin que esto afecte o altere a los programas de aplicación. El soporte para bases de datos distribuidas significa que una colección arbitraria de relaciones, bases de datos corriendo en una

mezcla de distintas máquinas y distintos sistemas operativos y que esté conectada por una variedad de redes, pueda funcionar como si estuviera disponible como en una única base de datos en una sola máquina."

12. *Regla de la no subversión*: "Si el sistema tiene lenguajes de bajo nivel, estos lenguajes de ninguna manera pueden ser usados para violar la integridad de las reglas y restricciones expresadas en un lenguaje de alto nivel (como SQL)."

2.7.Modelos arquitectónicos.

Con el desarrollo de grandes sistemas informáticos, el diseño no sólo implicó nuevas metodologías de algoritmos y estructuras de datos, el diseño y especificación global del sistema es un nuevo tipo de problema. Se necesita entonces, un modelo que disponga de manera conjunta y ordenada los elementos de software y hardware para cumplir una determinada función, este modelo recibe el nombre de modelo arquitectónico de software.

En un sentido amplio podemos decir que la arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene las siguientes características:

- Definir los módulos principales.
- Definir las responsabilidades que tendrá cada uno de estos módulos.
- Control, flujo de datos y secuencia de la información.
- Protocolos de interacción y comunicación.
- Ubicación en el hardware.

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La definición oficial de Arquitectura de Software de la IEEE (Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos) es: "La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución".

2.7.1. Arquitectura Cliente Servidor

El modelo cliente-servidor es un modelo en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el dialogo o solicita los recursos y servidor al proceso que responde a las solicitudes. Actualmente a esta arquitectura se le conoce como modelo de 2 capas.

Las características del modelo cliente servidor pueden ser resumidas en los siguientes puntos:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.

- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

Un ejemplo gráfico del modelo cliente servidor se muestra en la figura 1.15.

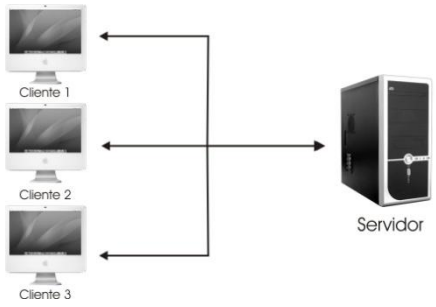


Figura 1.15 Modelo cliente – servidor

La estructura cliente-servidor nació el día en que alguien conecto una PC a una máquina Linux, nadie lo llamo entonces arquitectura de 2 capas, ya que no existía ninguna mas.

2.7.2. Arquitectura de 3 niveles.

La estrategia inicial del desarrollo de sistemas fue crear aplicaciones compactas, como resultado se obtuvo una gran cantidad de problemas de integración en sistemas de software complejos como pueden ser los sistemas de gestión de información integrados en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad e integración.

Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas, que normalmente son tres: una capa que servirá para guardar los datos (base de datos), una capa para centralizar la lógica de negocio (modelo) y por último una interfaz gráfica que facilite al usuario el uso del sistema. La figura 1.16 muestra una representación grafica de este modelo.

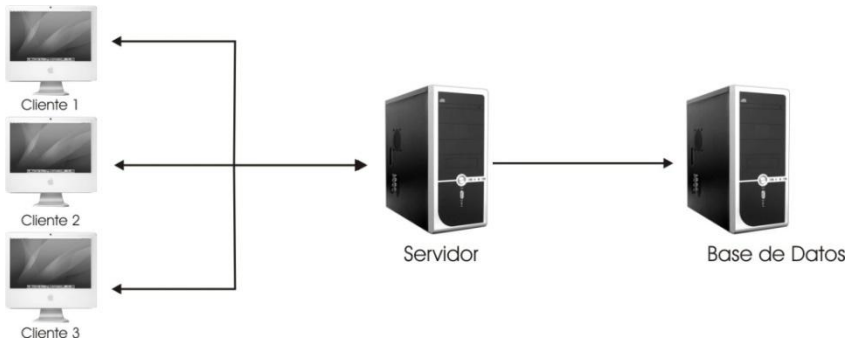


Figura 1.6 Arquitectura de 3 capas.

2.7.3. Plataforma WEB

Las aplicaciones basadas en Web son una implementación particular de la arquitectura cliente servidor. En los inicios del modelo cliente – servidor, cada aplicación tenía su propia interfaz cliente que debía ser instalada en las computadoras personales para cada usuario. Cada mejora en esta interfaz requería actualizar cada una de las computadoras en donde se encontraba instalado.

Las aplicaciones Web usan un software cliente en común denominado navegador (en inglés navigator o Web browser) el cual se comunica con un servidor Web para obtener información y ser presentada al usuario. Sin embargo, para que fuera posible utilizar el mismo cliente bajo diferente tipo de aplicaciones, se requirió estandarizar dos aspectos fundamentales en la arquitectura cliente – servidor:

- Definir la sintaxis y semántica que utilizan los elementos de software para comunicarse (servidor y cliente).
- Definir la sintaxis y semántica de la información (datos) que se envía al cliente y que debe ser presentada al usuario.

Protocolo HTTP

Un protocolo es un conjunto de estándares que controlan la secuencia de mensajes que ocurren en una comunicación entre entidades distintas que conforman una red. El protocolo http (HyperText Transfer Protocol, Protocolo para la Transferencia de HiperTexto) es el protocolo de comunicación usado para las transacciones Web. Es un protocolo orientado a transacciones y sigue el esquema de petición-respuesta entre un cliente y un servidor. A la información transmitida se la llama recurso y se la identifica mediante un URL (Universal Resource Identifier, Identificador Universal de Recurso). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

El protocolo http es un protocolo sin conexión y sin estado, es decir que después de que el servidor ha respondido la petición del cliente se rompe la conexión entre ambos. El desarrollo de

aplicaciones Web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente.

HTTP utiliza tipos MIME (Multipart Internet Mail Extension, Extensiones de Correo Internet Multipropósito) para la determinación de los tipos de dato que transporta. Cuando un servidor HTTP transmite información de vuelta a un cliente, incluye una cabecera que le indica al cliente sobre los tipos de dato que componen el documento. De la gestión de esos datos se encargan las utilidades que tenga el cliente (visor de imágenes, de vídeo, etc.)

Documentos HTML y World Wide Web

La estructura de la información que se muestra al usuario en el navegador web, se organiza en bloques de distinto contenido conectados a través de una serie de enlaces cuya activación o selección provoca la recuperación de información, es decir, los datos mostrados van más allá del hecho de mostrar caracteres en pantalla al usuario, siendo posible la interacción del usuario con la información a través de diversos elementos en pantalla, esta forma de comunicación recibe el nombre de hipertexto.

HTML (HyperText Markup Language, Lenguaje de Marcado de HiperTexto) es un lenguaje de marcado o codificación de texto usado para describir la estructura y contenido de información de una página web. HTML también puede describir hasta cierto punto la apariencia de una página WEB.

Los lenguajes de marcado suelen confundirse con lenguajes de programación, sin embargo, estos no son lo mismo, ya que el lenguaje de marcado no tiene las funciones aritméticas que tienen los lenguajes de programación como las variables. Históricamente, el marcado se usaba y se usa en la industria editorial y de la comunicación, así como entre autores, editores e imprentas.

World Wide Web es un sistema hipermedia interactivo desarrollado sobre Internet. La idea de hipermedia es la de juntar texto, imágenes, audio y vídeo dentro de un mismo envoltorio llamado documento. WWW se asienta sobre el protocolo HTTP (Hyper Text Transfer Protocol) y sobre el lenguaje de definición de documentos hipermedia HTML (HyperText Markup Language).

Servidor WEB.

Un servidor web es un programa que implementa el protocolo HTTP para transferir documentos web. El servidor web se mantiene en ejecución de manera constante a la espera de peticiones por parte de un cliente (navegador web) y responde a estas peticiones adecuadamente.

Un servidor web se mantiene a la espera de peticiones HTTP por parte de un cliente HTTP que solemos conocer como navegador. El cliente realiza una petición al servidor y éste le responde con el contenido que el cliente solicita.

Cuando se realiza una petición por medio de un cliente (navegador web) el servidor responde al cliente enviando el código HTML de la página; el cliente, una vez recibido el código, lo interpreta y lo exhibe en pantalla. El cliente es el encargado de interpretar el código HTML; el servidor tan sólo se limita a transferir el código de la página sin llevar a cabo ninguna interpretación de la misma.

Las aplicaciones web suelen ser la mejor opción en aplicaciones cliente – servidor, la razón es que al ejecutarse ésta en el servidor y no en la máquina del cliente, éste no necesita ninguna capacidad especial, cualquier cliente dotado de un navegador web básico puede utilizar este tipo de aplicaciones

3. DEFINICIÓN DE REQUERIMIENTOS

3.1.Introducción.

La definición de requerimientos, es la etapa más importante en el desarrollo del software, ya que en ella se reúnen todas las necesidades del usuario. Para lograr dicho objetivo se deben programar reuniones en las que se expone el problema a resolver, se toman en cuenta los puntos de vista y se llega a un acuerdo sobre la solución. Dicha solución debe delimitar el objetivo del proyecto es decir, definir el alcance. Durante estas reuniones, no se debe ahondar en detalles técnicos que puedan confundir al usuario, lo que se sugiere es utilizar herramientas gráficas como presentaciones las cuales muestren al usuario que se ha entendido claramente su necesidad, igualmente se pueden incluir diagramas UML de nivel 0 los cuales no son detallados y muestran las generalidades de la solución a nivel conceptual.

Durante el proceso de definición de requerimientos, el analista debe entender claramente el problema que se le presenta para hacer un estudio de viabilidad de la solución, para facilitar dicha actividad el analista debe comprender el ambiente laboral del usuario y preocuparse por ofrecer una herramienta amigable de acuerdo a la tecnología disponible y los gastos que se puedan generar. Para clarificar el tipo de solución ofrecida, se puede optar por mostrarle al usuario un demo donde se implemente un requerimiento sencillo, esto ayudará a reducir la resistencia al cambio y así obtener el apoyo del usuario ya que este se dará cuenta que la solución ofrecida es justo lo que necesita.

3.2.Planteamiento del Problema.

El centro Acasulco es un lugar donde se ofrece ayuda a personas con problemas de adicciones a alcohol o drogas, este lugar cuenta con un programa llamado “Programa de Satisfactores Cotidianos”, en el cual entran personas que demuestran presentar una dependencia alta a cualquiera de las sustancias, esto se define mediante pruebas psicológicas. Cabe señalar, que estas pruebas se van aplicando a lo largo de cuatro etapas del programa, donde se mide el nivel de dependencia, la adaptabilidad del individuo y la capacidad de eliminar por completo la dependencia. Es por eso que el principal problema a resolver es automatizar la calificación de pruebas psicológicas, ya que actualmente esta tarea se realiza de manera manual por los terapeutas con lo que se pierde tiempo y se presenta el riesgo del error humano el cual podría ocasionar dar un diagnostico erróneo.

Otro de los problemas identificados, es que no se tiene una forma de administrar la información de los pacientes (datos personales y expediente), ya que cada terapeuta decide qué y cómo documentarlo, lo cual hace una dependencia total al terapeuta que cuando este decide salir de la institución, hace muy difícil que alguno de sus compañeros tome su lugar.

3.3.Objetivo

Por lo anterior se definió que es necesario: Desarrollar un sistema que ayude en la automatización de asignación y calificación de pruebas psicológicas además de que pueda administrar la información personal y expediente de cada paciente. Este sistema debe cumplir con los siguientes objetivos:

- Ser una herramienta que apoye a la administración de pacientes y terapeutas.
- Automatizar el proceso de calificación de pruebas psicológicas.
- Reducir el tiempo de registro de pacientes y terapeutas.
- Reducir el error humano al dar un diagnóstico al paciente.
- Mejorar el tiempo de respuesta de los terapeutas.

3.4.Requerimientos

Los requerimientos de software, se clasifican en requerimientos funcionales y no funcionales. Un requerimiento funcional es aquel donde se muestra claramente la interacción entre el sistema y el usuario, en cambio un requerimiento no funcional no demuestra ninguna interacción con el usuario sin embargo es necesario para que el sistema funcione correctamente. Un ejemplo de requerimiento funcional es el siguiente: El terapeuta debe poder asignar las pruebas psicológicas a su paciente. En cambio un requerimiento no funcional puede ser el siguiente: El sistema debe guardar los datos capturados en una base de datos.

De acuerdo a la definición de requerimientos, se lograron identificar los siguientes según su tipo:

Requerimientos funcionales.

- El administrador dará de alta a todos los usuarios del sistema y gestionará el acceso al mismo.
- El terapeuta tendrá acceso a la información de cada uno de sus pacientes. Si un terapeuta deja la institución el administrador podrá reasignar al paciente.
- El administrador tendrá acceso a toda la información de pacientes y terapeutas.
- El terapeuta podrá programar una prueba psicológica a su paciente.
- El paciente deberá contestar la prueba mediante una interfaz del sistema.
- El terapeuta debe poder revisar el estatus y los resultados de las pruebas de sus pacientes.
- El terapeuta podrá programar sesiones de terapia con el paciente
- El terapeuta podrá capturar y/o guardar en el expediente del paciente el resultado de cada terapia
- El terapeuta podrá diseñar y dar de alta nuevas pruebas psicológicas.
- El terapeuta podrá consultar e ingresar documentos electrónicos al expediente del paciente.

- Todos los usuarios se deberán autentificar mediante una clave y contraseña.
- Los usuarios podrán imprimir cualquier información que puedan consultar en el sistema.

Requerimientos no funcionales.

- El sistema debe ser accesible desde intranet
- El sistema debe mostrar una interfaz web.
- El sistema deberá guardar la información en una base de datos a excepción de los expedientes.
- El sistema deberá guardar el expediente del paciente en un directorio nombrado con su clave en el sistema operativo.
- El sistema deberá generar la clave de usuario al concatenar lo siguiente: 2 letras de apellido paterno, 2 letras de apellido materno, 2 letras de primer nombre y fecha de nacimiento en formato AAMMDD, donde AA es año en 2 dígitos, MM es mes y DD es día. Ejemplo: Ramírez Alcalde Georgina, 1981-02-03: RAALGE810203.

4. ANÁLISIS.

4.1.Introducción

Definir los requerimientos que debe satisfacer el sistema nos ayuda a comprender cuál es el problema que se desea resolver. Resulta erróneo pensar que los requerimientos no son más que documentación tediosa, al contrario, resultan esenciales para realizar un diseño modular y de alta calidad.

La siguiente etapa en la construcción de un proyecto de software es el Análisis. Esta etapa pretende responder a las preguntas: ¿Qué debe hacerse? Y ¿Qué sistema deseamos construir?

4.2.Análisis de Requisitos

El análisis de requisitos es un proceso que busca especificar las características operacionales: función y datos. Se busca establecer interfaces con otros elementos que involucran la parte del problema y estableces fronteras en cuanto a la funcionalidad del sistema se refiere.

Aunque en principio el análisis de requerimientos parece una tarea sencilla, en realidad no lo es, ya que la información suministrada por el usuario puede resultar densa y con conceptos no triviales para un analista de sistemas, de este modo resulta indispensable mantener comunicación directa y constante con el usuario, con el fin de evitar ambigüedades o una mala interpretación que puede llevar a la implementación de una solución a un problema que nunca existió.

Tanto el cliente como el analista juegan un papel activo en el análisis de requerimientos, el cliente debe expresar sus necesidades en torno a la lógica del negocio que maneja, el analista, sin embargo, actúa como interrogador, como consultor y como persona que resuelve problemas.

Para crear un puente entre el desarrollo de la solución y el usuario, el analista debe crear modelos de flujos de información y de control que sean conceptualmente claros donde se describa cómo se usará el sistema.

El análisis de requerimientos permite al analista representar el dominio de la información que será tratada por el programa. El análisis de requerimientos da al diseñador la representación de la información y las funciones que pueden ser traducidas en datos, arquitectura y diseño procedimental. La especificación de requerimientos suministra al técnico y al cliente, los medios para valorar la calidad de los programas, una vez que se haya construido. Estas tareas finalizan con el visto bueno del usuario para formalizar y dejar en claro que se está de acuerdo con la solución ofrecida.

4.3.Casos de uso.

Tras recompilar los requisitos que debe cumplir el sistema, el siguiente paso es describir como se usará el sistema. Esta descripción se realiza mediante diagramas conocidos como casos de uso.

Los diagramas de caso de uso representan cómo opera el usuario en el sistema, además de la forma y orden de cómo interactúan los elementos. Estos diagramas describen las acciones y las reacciones del sistema desde el punto de vista del usuario, son descripciones de la funcionalidad independientes de la implementación.

El procedimiento para generar casos de uso es el siguiente:

- Identificar las personas (o dispositivos) que utilizarán el sistema (Actores).
- Identificar las funciones que realiza un actor.
- Identificar que actor notificará al sistema de los cambios en el entorno.
- Definir qué información necesita el actor del sistema.

Podemos identificar y clasificar a los actores de un sistema de la siguiente forma:

- Principales: Personas que usan el sistema, una misma persona física puede interpretar distintos papeles de actores, la denominación “actor” describe el rol desempeñado.
- Secundarios: Personas que mantienen o administran el sistema.
- Otros Sistemas y dispositivos: Sistemas y dispositivos con los que el sistema interactúa.

Los casos de uso se determinan observando y precisando actor por actor las secuencias de interacción y escenarios desde el punto de vista del usuario. Un escenario es una instancia de caso de uso, es decir, el modelado con datos específicos. Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso.

De acuerdo a los requerimientos del usuario, el diagrama de Casos de Uso de la figura 3.1 muestra la funcionalidad que cubrirá lo solicitado:

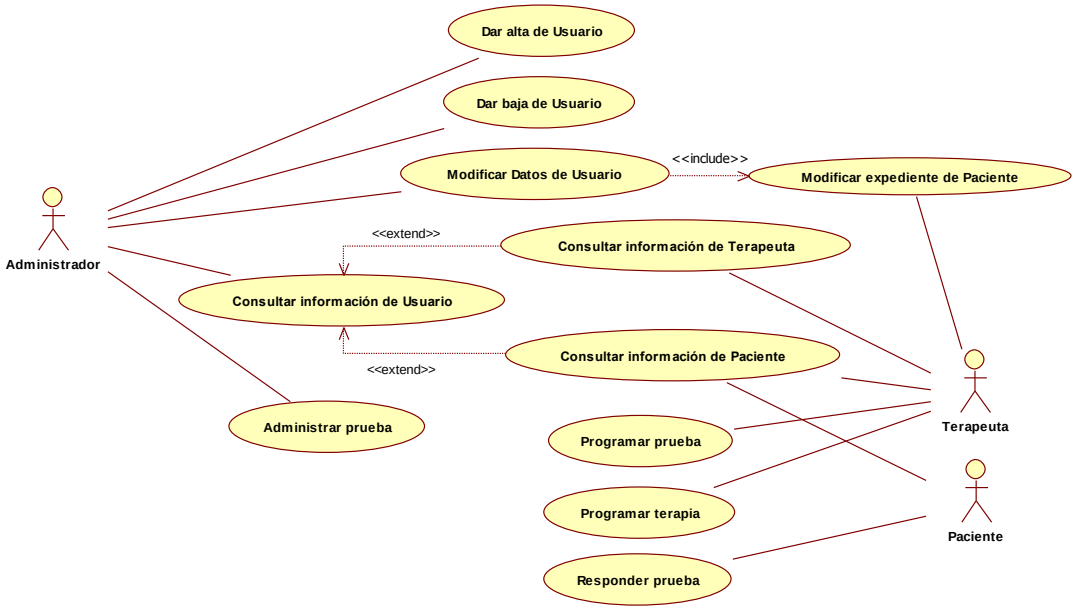


Figura 3.1 Diagrama de casos de uso

De acuerdo al diagrama, se tiene un total de 8 Casos de Uso principales, para describir a cada uno de ellos, se programaron sesiones de trabajo con el usuario dando como resultado su especificación como se muestra a continuación:

CU 1	Dar alta de Usuario	
Meta y Contexto	Dar de alta al usuario en el sistema	
Precondiciones	Que se cuente con los datos del Usuario Que el Administrador este autentificado en el sistema	
Condición de Éxito	Que el Usuario esté dado de alta en el sistema y que tenga acceso de acuerdo a su perfil	
Condiciones de Fallo	Que no exista el Usuario dado de alta en el sistema	
Actor Principal, Actor(es) Secundarios	Administrador Usuario (Terapeuta o Paciente), Sistema	
Disparador	Se le pide al Administrador que realice el alta un nuevo Usuario	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige el tipo de Usuario (Terapeuta o Paciente)
	2	El Administrador elige como tipo de Usuario al Terapeuta
	3	El Administrador captura los datos del Terapeuta
	4	El Sistema genera la clave de usuario de acuerdo a lo siguiente: El Sistema concatena: primeras 2 letras del apellido paterno, primeras 2 letras del apellido materno (en caso de no contar con este, “XX”, primeras 2 letras del nombre y fecha de nacimiento con el siguiente formato AAMMDD, donde AA es año en dos dígitos, MM es mes y DD día.
	5	El Sistema pide que se defina la contraseña pidiendo confirmación

	6	El Administrador pide al Usuario que escriba y confirme la contraseña
	7	El Sistema muestra la confirmación de que el Usuario fue creado correctamente
Flujos Alternativos	**Paso**	**Acción Ramificada**
	1	El Administrador elige como tipo de Usuario al Paciente: 1. El Administrador captura los datos del Paciente 2. El Administrador elige al Terapeuta asignado al Paciente 3. Ir al paso 4 del escenario principal
	4	El Sistema detecta que ya existe un usuario con esta clave: 1. El Sistema muestra los datos del usuario con dicha clave 2. El Administrador verifica si se trata del mismo Usuario 3. El Administrador confirma que se trata del mismo Usuario 4. Fin del caso de uso
	4.2	No se trata del mismo Usuario: 1. El Administrador genera manualmente la clave 2. Ir al paso 6 del escenario principal
Variaciones	**Descripción**	
	N/A	

Los casos de uso restantes están en el anexo A adjunto.

4.4.Modelado

El modelado del sistema tiene la función de ayudar al analista a entender la información función y comportamiento del sistema. Los diferentes documentos de modelado ayudan a determinar la completitud, consistencia y precisión de los requerimientos además de sentar la base para el diseño.

El modelado de un sistema se realiza principalmente de dos formas: El modelo funcional y el modelo del comportamiento.

Modelo Funcional: El modelo funcional se realiza indicando tres etapas básicas: Entrada, procesamiento y salida. El modelo comienza por un diagrama sencillo de contexto para posteriormente realizar iteraciones hasta llegar a niveles más detallados de funcionalidad. Como ejemplo podemos mencionar el Diagrama de Flujo de Datos (DFD).

Modelo de Comportamiento: El modelo de comportamiento describe las características del sistema en los términos estimulo – respuesta. Representa los estados del software y los sucesos que causan que cambie de estado, estos diagramas se denominan comúnmente diagramas de estado.

4.4.1. Diagrama de Flujo de Datos.

El diagrama de flujo de datos (DFD) es una representación gráfica del "flujo" de datos a través de un sistema de información, pretende mostrar la transformación de la información a medida que fluye por el sistema.

El modelo de DFD comienza por un nivel 0, conocido también como modelo de contexto, que muestra al sistema como una burbuja con datos de entrada y salida. Con un diagrama de flujo de datos, los usuarios van a poder visualizar la forma en que el sistema funciona, lo que el sistema va a lograr, y cómo el sistema se pondrá en práctica.

La manera en que cualquier sistema es desarrollado puede determinarse a través de un diagrama de flujo de datos. El desarrollo de un DFD ayuda en la identificación de los datos de la transacción en el modelo de datos, que describiremos más adelante. La figura 3.2 muestra como ejemplo el diagrama de flujo de datos nivel 0 de nuestra aplicación.

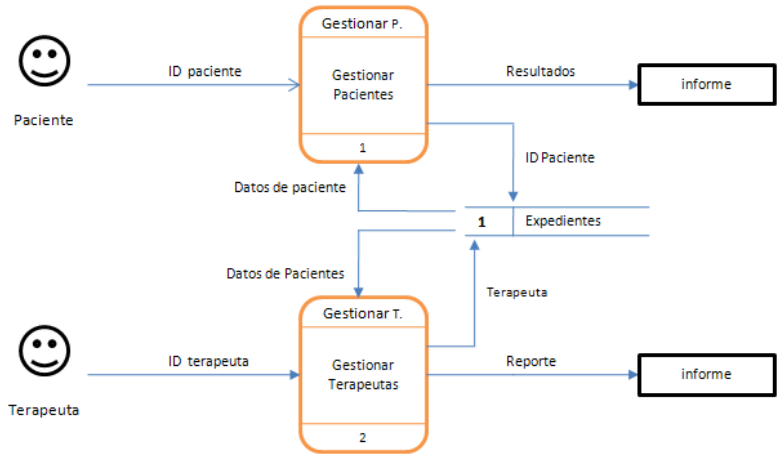


Figura 3.2 Diagrama de Flujo de Datos Nivel 0.

De la misma manera, la figura 3.3 muestra el diagrama de flujo de datos de nivel 1 de la aplicación.

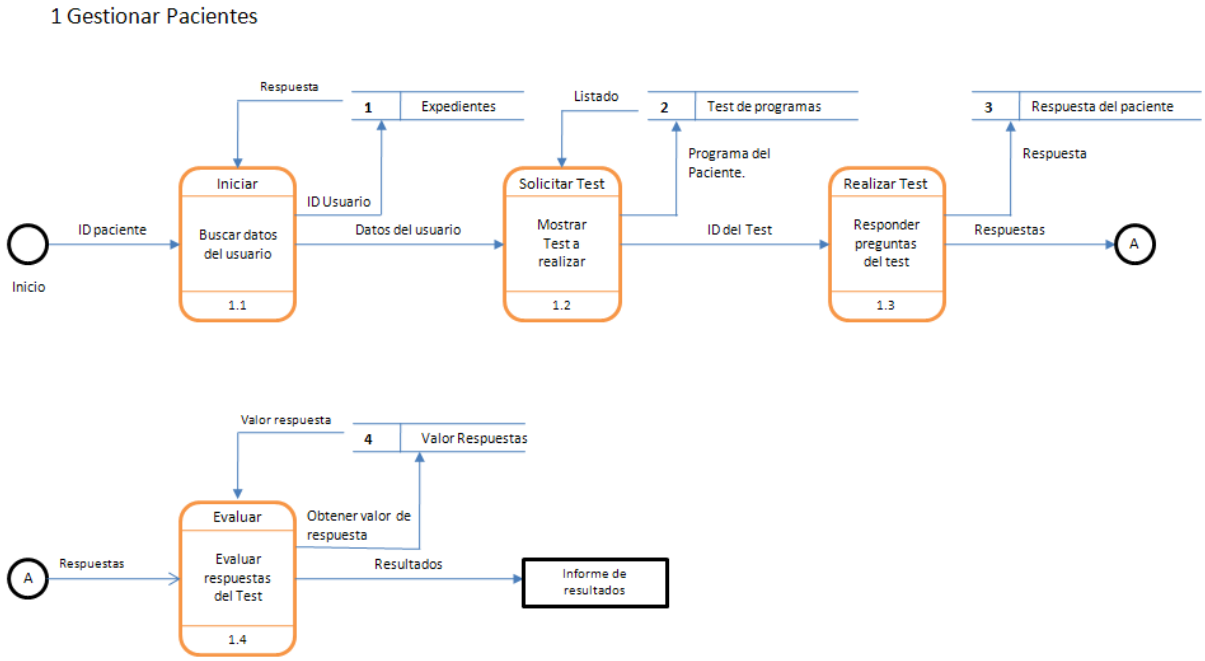


Figura 3.3 Diagrama de Flujo de Datos Nivel 1

4.4.2. Diagrama de Estados.

Los diagramas de estado son representaciones gráficas que muestran el comportamiento del sistema. Muestran los estados del sistema y los sucesos que causan que cambie de estado, nos ayudan a identificar los flujos de control de cada proceso mostrando la manera en cómo se mueve el sistema de un estado a otro.

La figura 3.4 muestra como ejemplo el diagrama de estados de nuestra aplicación.

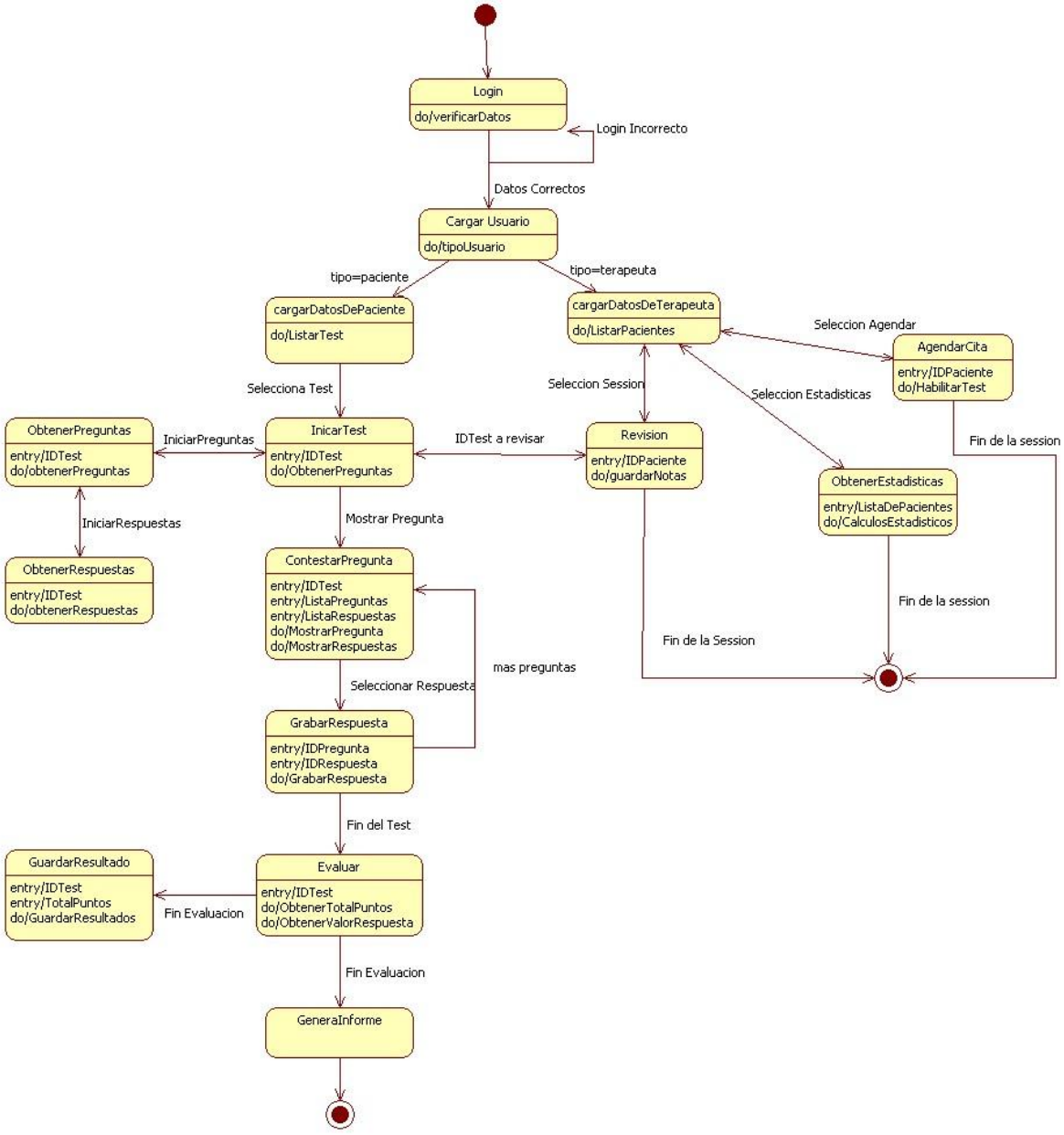


Figura 3.4 Diagrama de Estados

4.4.3. Diagrama de Actividades.

El diagrama de actividades es una representación grafica del flujo de acciones que se necesitan realizar para llevar a cabo la tarea de un caso de uso. Podemos considerar estos diagramas como diagramas de flujo que representan la lógica de negocio que debe seguir nuestro sistema.

El diagrama debe contener las reglas y condiciones que se deben tomar en cuenta en un caso de uso. La figura 3.5 muestra el diagrama de actividades para el caso de uso “Responder prueba”.

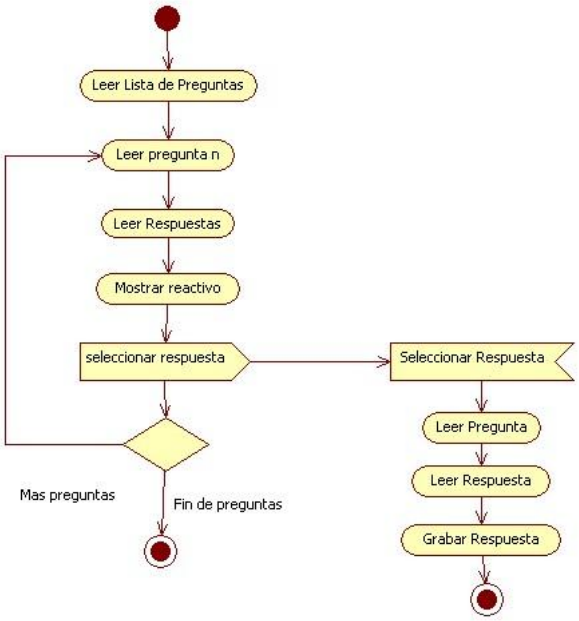


Figura 3.5 Diagrama de Actividades

5. DISEÑO

5.1.Introducción

El diseño es el primer paso de la fase de desarrollo de cualquier producto o sistema de ingeniería. La finalidad del diseño es generar la documentación suficiente que permita la construcción del sistema en cuestión.

El diseño responde a la pregunta ¿Cómo debe hacerse? y ¿Cómo deseamos construir el sistema? Para ello habrá que contemplar recursos y tiempos, que al final serán traducidos en costos del sistema.

El diseño debe satisfacer todos los requisitos planteados en el análisis, contemplar las reglas que deben seguir los procesos de información así como la forma en que los resultados serán presentados al usuario.

La fase de diseño puede definirse como: el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física."

5.2.Diseño de la Base de Datos.

El diseño y modelado de datos es una de las actividades más importantes en el desarrollo de sistemas, ya que en este se describen las estructuras desde el punto de vista lógico y físico, además de que se documenta el diccionario de datos. Enseguida se muestran los temas que abarcan el Diseño de la Base de Datos del Sistema.

5.2.1. Modelo Lógico de la Base de Datos

El Modelo Lógico se representa a partir de un diagrama entidad relación (DER), en este se describe la estructura que guardarán los datos de interés para las actividades de los usuarios. El Modelo Lógico se centra en la información independiente del procesamiento del sistema.

Los datos se agrupan en entidades además de que se muestra la relación entre ellas, a una entidad también se le denomina como objeto de datos. Un objeto de datos únicamente encapsula datos, por esta razón no representa operaciones.

De acuerdo a las sesiones de trabajo con el usuario, se determinaron las relaciones y entidades que describen a las actividades que realizan. Como resultado de ello, se determinó el siguiente diagrama de la figura 4.1 como Modelo Lógico de Datos:

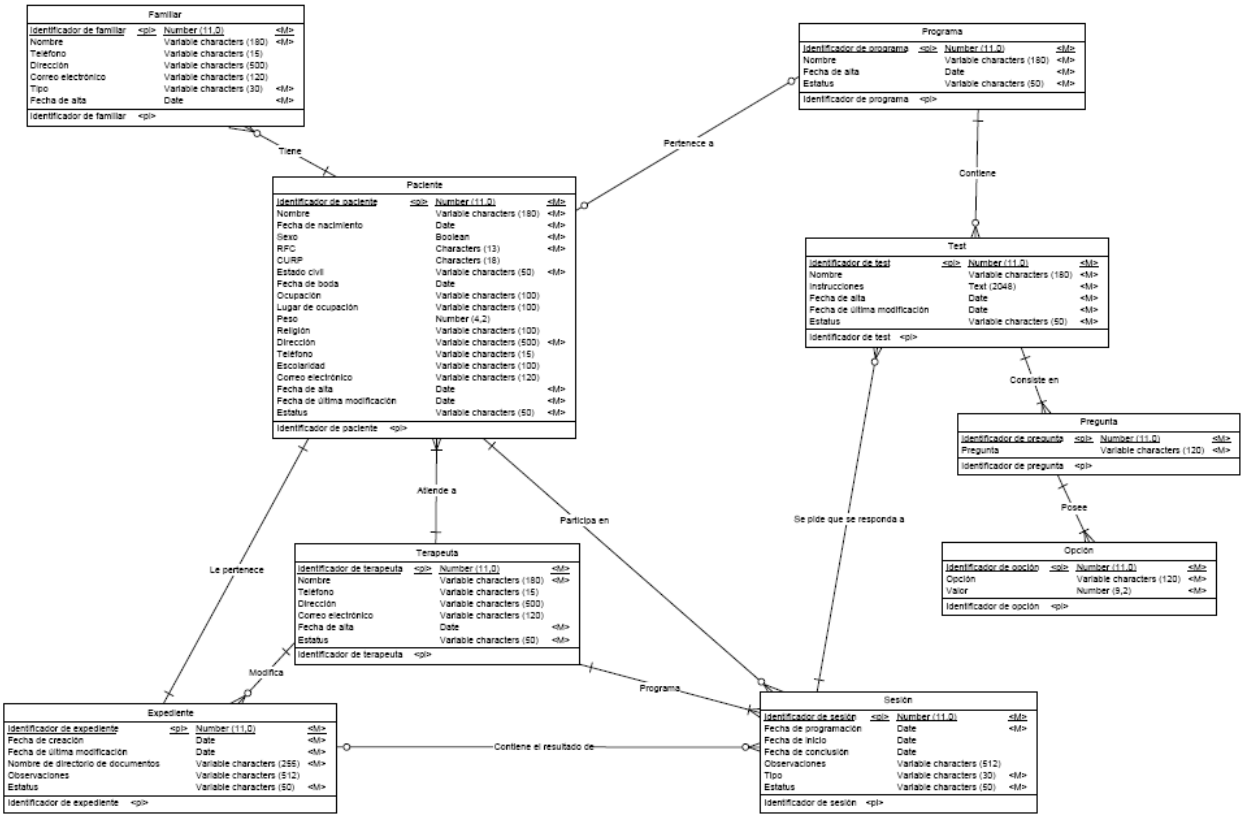
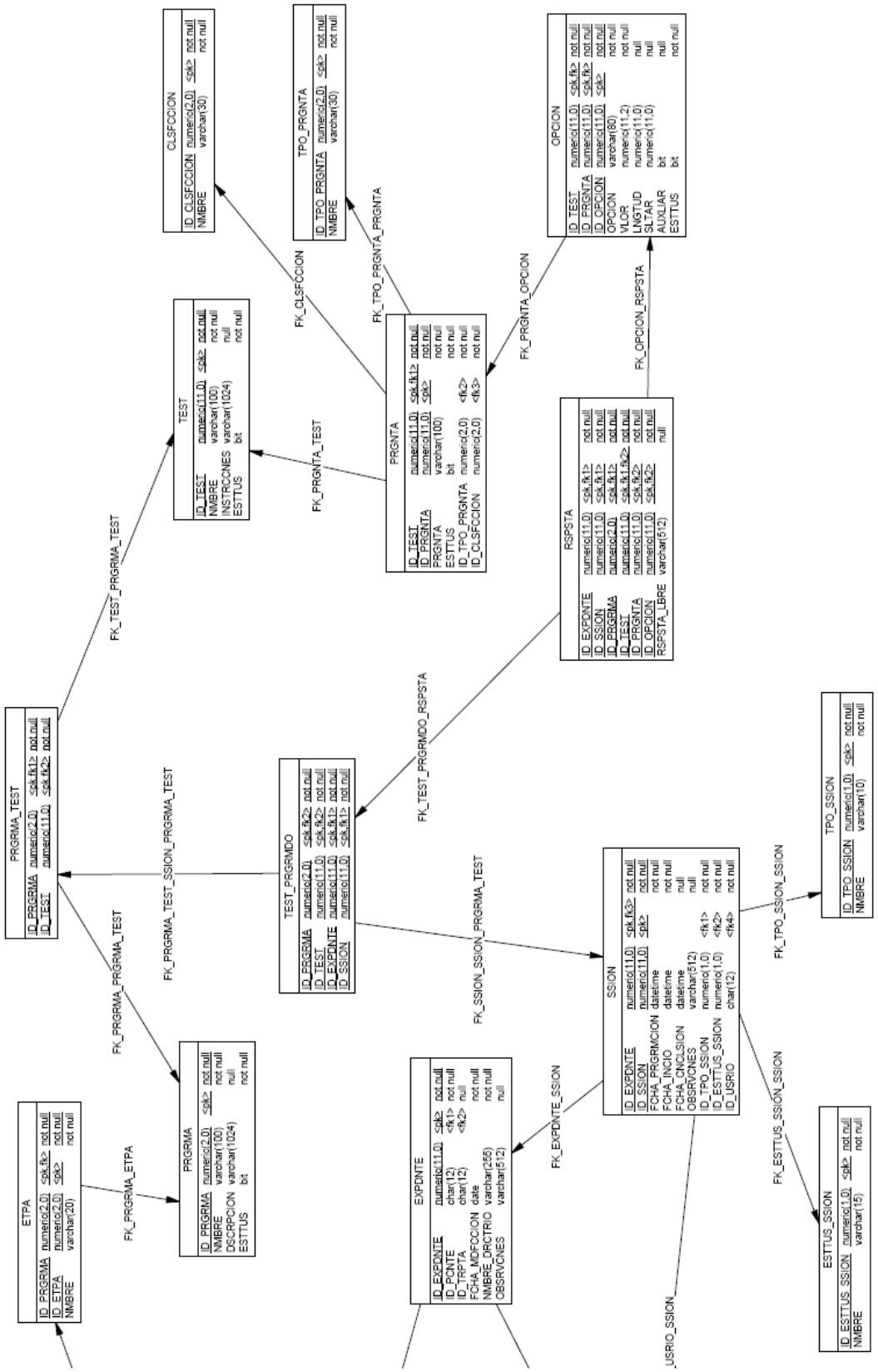
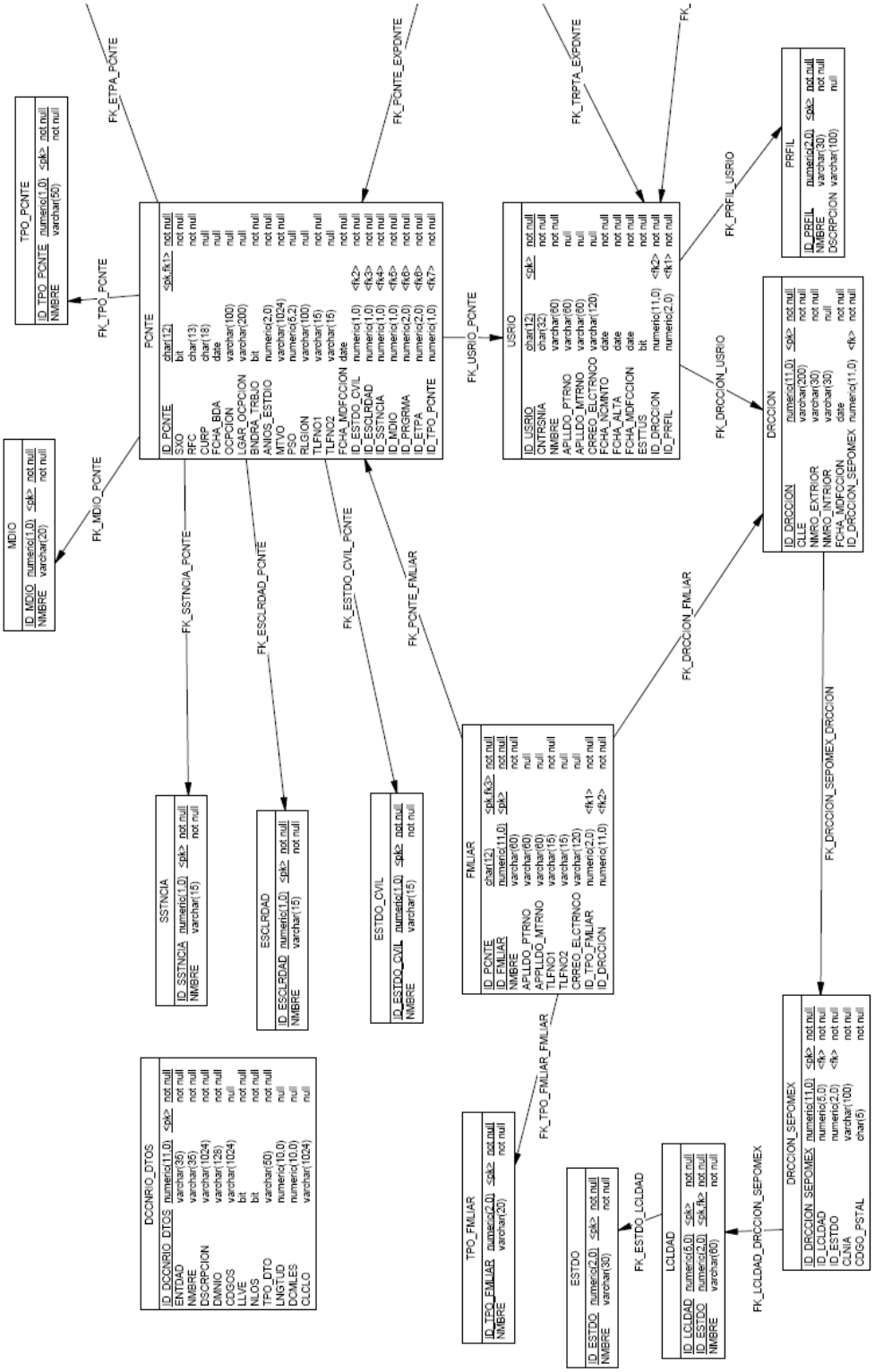


Figura 4.1 Modelo Lógico de Datos

5.2.2. Modelo Físico de la Base de Datos

El Modelo Físico de datos sirve a los desarrolladores y administradores del sistema, para entender y modificar la estructura de la base de datos, la diferencia entre el Modelo Físico y Lógico, es que en el Modelo Físico se describen además de las características de las actividades del negocio que se van a representar, datos auxiliares y administrativos mismos del sistema, a continuación se muestra el diagrama Entidad Relación del Modelo Físico de Datos.


CLSFCCION
ID_CLSFCCION numerico(2,0) <pk> not null
NMBRE varchar(30) not null
TPO_PRGNTA
ID_TPO_PRGNTA numerico(2,0) <pk> not null
NMBRE varchar(30) not null
OPCION
ID_TEST numerico(11,0) <pk,fk> not null
ID_PRGNTA numerico(11,0) <pk,fk> not null
ID_OPCION numerico(11,0) <pk> not null
OPCION varchar(80) not null
VLOR numerico(11,2) not null
LNGTUD numerico(11,0) null
SLTAR numerico(11,0) null
AUXLIAR bit null
ESTTUS bit not null
TEST
ID_TEST numerico(11,0) <pk> not null
NMBRE varchar(100) not null
INSTRCCNES varchar(1024) null
ESTTUS bit not null
PRGNTA
ID_TEST numerico(11,0) <pk,fk1> not null
ID_PRGNTA numerico(11,0) <pk> not null
PRGNTA varchar(100) not null
ESTTUS bit not null
ID_TPO_PRGNTA numerico(2,0) <fk2> not null
ID_CLSFCCION numerico(2,0) <fk3> not null
RSPSTA
ID_EXPDNTE numerico(11,0) <pk,fk1> not null
ID_SSION numerico(11,0) <pk,fk1> not null
ID_PRGRMA numerico(2,0) <pk,fk1> not null
ID_TEST numerico(11,0) <pk,fk1,fk2> not null
ID_PRGNTA numerico(11,0) <pk,fk2> not null
ID_OPCION numerico(11,0) <pk,fk2> not null
RSPSTA_LBRE varchar(512) null
PRGRMA_TEST
ID_PRGRMA numerico(2,0) <pk,fk1> not null
ID_TEST numerico(11,0) <pk,fk2> not null
TEST_PRGRMDO
ID_PRGRMA numerico(2,0) <pk,fk2> not null
ID_TEST numerico(11,0) <pk,fk2> not null
ID_EXPDNTE numerico(11,0) <pk,fk1> not null
ID_SSION numerico(11,0) <pk,fk1> not null
ETPA
ID_PRGRMA numerico(2,0) <pk,fk> not null
ID_ETPA numerico(2,0) <pk> not null
NMBRE varchar(20) not null
PRGRMA
ID_PRGRMA numerico(2,0) <pk> not null
NMBRE varchar(100) not null
DSCRPCION varchar(1024) null
ESTTUS bit not null
EXPDNTE
ID_EXPDNTE numerico(11,0) <pk> not null
ID_PDNTE char(12) <fk1> not null
ID_TRPTA char(12) <fk2> null
FCHA_MDFCCION date not null
NMBRE_DRCTRIO varchar(255) not null
OBSRVCNES varchar(512) null
SSION
ID_EXPDNTE numerico(11,0) <pk,fk3> not null
ID_SSION numerico(11,0) <pk> not null
FCHA_PRGRMCION datetime not null
FCHA_INCIO datetime not null
FCHA_CNCLSION datetime null
OBSRVCNES varchar(512) null
ID_TPO_SSION numerico(1,0) <fk1> not null
ID_ESTTUS_SSION numerico(1,0) <fk2> not null
ID_USRIO char(12) <fk4> not null
TPO_SSION
ID_TPO_SSION numerico(1,0) <pk> not null
NMBRE varchar(10) not null
ESTTUS_SSION
ID_ESTTUS_SSION numerico(1,0) <pk> not null
NMBRE varchar(15) not null
FK_CLSFCCION
FK_TPO_PRGNTA_PRGNTA
FK_PRGNTA_OPCION
FK_OPCION_RSPSTA
FK_TEST_PRGRMA_TEST
FK_PRGRMA_TEST_SSION_PRGRMA_TEST
FK_PRGNTA_TEST
FK_TEST_PRGRMDO_RSPSTA
FK_PRGRMA_PRGRMA_TEST
FK_SSION_SSION_PRGRMA_TEST
FK_PRGRMA_ETPA
FK_EXPDNTE_SSION
FK_TPO_SSION_SSION
FK_ESTTUS_SSION_SSION
USRIO_SSION


TPO_PONTE
ID_TPO_PONTE numeric(1,0) <pk> not null
NMBRE varchar(50) not null
MDIO
ID_MDIO numeric(1,0) <pk> not null
NMBRE varchar(20) not null
PONTE
ID_PONTE char(12) <pk,fk1> not null
SXO bit not null
RFC char(13) not null
CURP char(18) null
FCHA_BDA date null
OCPCION varchar(100) null
LGAR_OCPCION varchar(200) null
BNDRA_TRBJO bit not null
ANIOS_ESTDIO numeric(2,0) not null
MTVO varchar(1024) not null
PSO numeric(6,2) null
RLGION varchar(100) null
TLFNO1 varchar(15) not null
TLFNO2 varchar(15) null
FCHA_MDFCCION date not null
ID_ESTDO_CVIL numeric(1,0) <fk2> not null
ID_ESCLRDAD numeric(1,0) <fk3> not null
ID_SSTNCIA numeric(1,0) <fk4> not null
ID_MDIO numeric(1,0) <fk5> not null
ID_PRGRMA numeric(2,0) <fk6> not null
ID_ETPA numeric(2,0) <fk6> not null
ID_TPO_PONTE numeric(1,0) <fk7> not null
USRIO
ID_USRIO char(12) <pk> not null
CNTRSNIA char(32) not null
NMBRE varchar(60) not null
APLLDO_PTRNO varchar(60) null
APLLDO_MTRNO varchar(60) null
CRREO_ELCTRNCO varchar(120) null
FCHA_NCMNTO date not null
FCHA_ALTA date not null
FCHA_MDFCCION date not null
ESTTUS bit not null
ID_DRCCION numeric(11,0) <fk2> not null
ID_PRFIL numeric(2,0) <fk1> not null
PRFIL
ID_PRFIL numeric(2,0) <pk> not null
NMBRE varchar(30) not null
DSCRPCION varchar(100) null
DRCCION
ID_DRCCION numeric(11,0) <pk> not null
CLLE varchar(200) not null
NMRO_EXTRIOR varchar(30) not null
NMRO_INTRIOR varchar(30) null
FCHA_MDFCCION date not null
ID_DRCCION_SEPOMEX numeric(11,0) <fk> not null
SSTNCIA
ID_SSTNCIA numeric(1,0) <pk> not null
NMBRE varchar(15) not null
ESCLRDAD
ID_ESCLRDAD numeric(1,0) <pk> not null
NMBRE varchar(15) not null
ESTDO_CVIL
ID_ESTDO_CVIL numeric(1,0) <pk> not null
NMBRE varchar(15) not null
FMLIAR
ID_PONTE char(12) <pk,fk3> not null
ID_FMLIAR numeric(11,0) <pk> not null
NMBRE varchar(60) not null
APLLDO_PTRNO varchar(60) null
APLLDO_MTRNO varchar(60) null
TLFNO1 varchar(15) not null
TLFNO2 varchar(15) null
CRREO_ELCTRNCO varchar(120) null
ID_TPO_FMLIAR numeric(2,0) <fk1> not null
ID_DRCCION numeric(11,0) <fk2> not null
DCCNRIO_DTOS
ID_DCCNRIO_DTOS numeric(11,0) <pk> not null
ENTDAD varchar(35) not null
NMBRE varchar(35) not null
DSCRPCION varchar(1024) not null
DMNIO varchar(128) not null
CDGOS varchar(1024) null
LLVE bit not null
NLOS bit not null
TPO_DTO varchar(50) not null
LNGTUD numeric(10,0) null
DOMLES numeric(10,0) null
CLCLO varchar(1024) null
TPO_FMLIAR
ID_TPO_FMLIAR numeric(2,0) <pk> not null
NMBRE varchar(20) not null
ESTDO
ID_ESTDO numeric(2,0) <pk> not null
NMBRE varchar(30) not null
LCLDAD
ID_LCLDAD numeric(6,0) <pk> not null
ID_ESTDO numeric(2,0) <pk,fk> not null
NMBRE varchar(60) not null
DRCCION_SEPOMEX
ID_DRCCION_SEPOMEX numeric(11,0) <pk> not null
ID_LCLDAD numeric(6,0) <fk> not null
ID_ESTDO numeric(2,0) <fk> not null
CLNIA varchar(100) not null
CDGO_PSTAL char(5) not null
FK_ETPA_PONTE
FK_TPO_PONTE
FK_PONTE_EXPONTE
FK_MDIO_PONTE
FK_USRIO_PONTE
FK_TRPTA_EXPONTE
FK_PRFIL_USRIO
FK_DRCCION_USRIO
FK_SSTNCIA_PONTE
FK_ESCLRDAD_PONTE
FK_ESTDO_CVIL_PONTE
FK_PONTE_FMLIAR
FK_DRCCION_FMLIAR
FK_DRCCION_SEPOMEX_DRCCION
FK_TPO_FMLIAR_FMLIAR
FK_ESTDO_LCLDAD
FK_LCLDAD_DRCCION_SEPOMEX

Como se puede notar, en el Modelo Físico se incluyen y normalizan todas las estructuras de datos necesarias para dar soporte a la funcionalidad del sistema que no necesariamente interesa al usuario, por lo tanto aumentan los datos y las entidades. Como parte del Modelo Físico, se incluyó una tabla para almacenar el diccionario de datos, y este podrá ser consultado para aclarar cualquier duda sobre el contenido de la base de datos y la forma en que se almacena y genera la información.

5.3. Diccionario de Datos.

Como se vio en el capítulo 1, es importante describir el contenido de los datos en un elemento llamado Diccionario de Datos, debido a las características físicas de los mismos que tienen que ver con el almacenamiento restricciones y organización, se agregaron algunas columnas extra a las presentadas en el capítulo 1. A continuación se presenta el diccionario de datos para una de las entidades del modelo físico (el diccionario de datos completo se documenta en el anexo D):

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
1	PCNTE	ID_PCNTE	Identificador del Paciente	[A-Z]{6}AAMMDD	N/A	X	No	char	12	N/A	Se calcula mediante las primeras dos letras del apellido paterno, dos del apellido materno, dos del primer nombre y su fecha de nacimiento en formato AAMMDD, donde AA es año, MM mes y DD dìa. En caso de que no cuente con alguno de los dos apellidos, se colocará "XX"
2	PCNTE	SXO	Identificador del sexo del Paciente	[0,1]	0=Masculino 1=Femenino		No	bit	N/A	N/A	N/A
3	PCNTE	RFC	Registro Federal de Contribuyentes del Paciente	[A-Z]{4}AAMMDD([0-9][A-Z][0-9])?	N/A		No	char	13	N/A	N/A
4	PCNTE	CURP	Clave Unica de Registro de Población del Paciente	[A-Z]{4}AAMMDD[A-Z]{6}[0-9]{2}	N/A		Si	char	18	N/A	N/A
5	PCNTE	FCHA_BDA	Fecha en la que se casó el Paciente, con ellos se calcula el número de años que lleva de casado, que es uno de los factores de análisis en el programa	AAAAMMDD	N/A		Si	date	N/A	N/A	Si no se cuenta con la fecha exacta, al menos se debe proporcionar el año y el mes y por default se pondrá el primer dìa del mes
6	PCNTE	OCPCION	Descripción de la ocupación del Paciente	Todos los valores	N/A		Si	varchar	100	N/A	N/A
7	PCNTE	LGAR_OCPCION	Descripción del sitio donde labora el Paciente	Todos los valores	N/A		Si	varchar	200	N/A	N/A
8	PCNTE	BNDRA_TRBJO	Indicador que permite saber si la actividad que realiza el Paciente es remunerativa	[0,1]	0=No es un trabajo remunerado 1=Si es un trabajo remunerado		No	bit	N/A	N/A	N/A
9	PCNTE	ANIOS_ESTDIO	Número de años máximos de estudio del Paciente de acuerdo a su grado de escolaridad	[0-9]{1-2}	N/A		No	numeric	2	0	N/A
10	PCNTE	MTVO	Descripción del motivo por el que el Paciente decidió entrar al programa	Todos los valores	N/A		No	varchar	1024	N/A	N/A
11	PCNTE	PSO	Número que indica el peso en kilogramos del	[0-9]{3}.[0-9]{2}	N/A		Si	numeric	5	2	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			Paciente								
12	PCNTE	RLGION	Descripción de la religión del Paciente	Todos los valores	N/A		Si	varchar	100	N/A	N/A
13	PCNTE	TLFNO1	Número telefónico principal del Paciente	Todos los valores	N/A		No	varchar	15	N/A	N/A
14	PCNTE	TLFNO2	Número telefónico secundario o auxiliar del Paciente que normalmente es el celular	Todos los valores	N/A		Si	varchar	15	N/A	N/A
15	PCNTE	FCHA_MDFCCION	Fecha de última modificación de datos del Paciente en el sistema	AAAAMMDD	N/A		No	date	N/A	N/A	Si cambia de valor en el sistema cualquiera de los siguientes campos, entonces se registra la fecha del sistema
16	PCNTE	ID_ESTDO_CVIL	Código del estado civil del Paciente	[1-9]	1=Soltero(a) 2=Casado(a) 3=Unión Libre 4=Viudo(a) 5=Divorciado(a)		No	numeric	1	0	N/A
17	PCNTE	ID_ESCLRDAD	Código del grado máximo de escolaridad del Paciente	[1-9]	1=Primaria 2=Secundaria 3=Bachillerato 4=Licenciatura 5=Maestría 6=Doctorado		No	numeric	1	0	N/A
18	PCNTE	ID_SSTNCIA	Código de la sustancia que consume el Paciente	[1-9]	1=Alcohol 2=Marihuana 3=Cocaina 4=Benzodicepinas 5=Barbituricos 6=Poliusuario		No	numeric	1	0	N/A
19	PCNTE	ID_MDIO	Código del medio por el que el Paciente se enteró del programa	[1-9]	1=Poster 2=Radio 3=TV 4=Servicios Médicos 5=Tríptico 6=Internet		No	numeric	1	0	N/A
20	PCNTE	ID_PRGRMA	Código del programa de ayuda psicológica en el que participa el Paciente	([1-9],[1-9][0-9])	1=Programa de Satisfactores Cotidianos		No	numeric	2	0	N/A
21	PCNTE	ID_ETPA	Código de la etapa en la que se encuentra el Paciente en el programa	([1-9],[1-9][0-9])	1=Primer Contacto 2=Admisión 3=Evaluación 1 4=Evaluación 2 5=Tratamiento 6=Seguimiento 1		No	numeric	2	0	N/A
22	PCNTE	ID_TPO_PCNTE	Código del tipo de Paciente que define el origen de donde proviene	[1-9]	1=Personal Administrativo de la UNAM 2=Estudiante de la UNAM 3=Público en General		No	numeric	1	0	N/A

Tabla 4.1 Diccionario de datos

5.4. Diagrama de clase

Una clase es la descripción de características y acciones que puede realizar un grupo de objetos con las mismas características, en otras palabras es la plantilla que caracteriza a cierto grupo de objetos. Gráficamente, las clases se representan en una caja rectangular divida en 3 compartimientos: en la parte superior se encuentra el nombre de la clase, en la parte media se encuentran los atributos y en la parte inferior se encuentran las operaciones o métodos de la clase. Las clases se relacionan entre sí a través de las relaciones, que gráficamente son líneas rectas que constan de roles y cardinalidad. Esta representación grafica de relaciones y clases forman el diagrama de clases. El diagrama de clases de la aplicación se muestra en la figura 4.3.

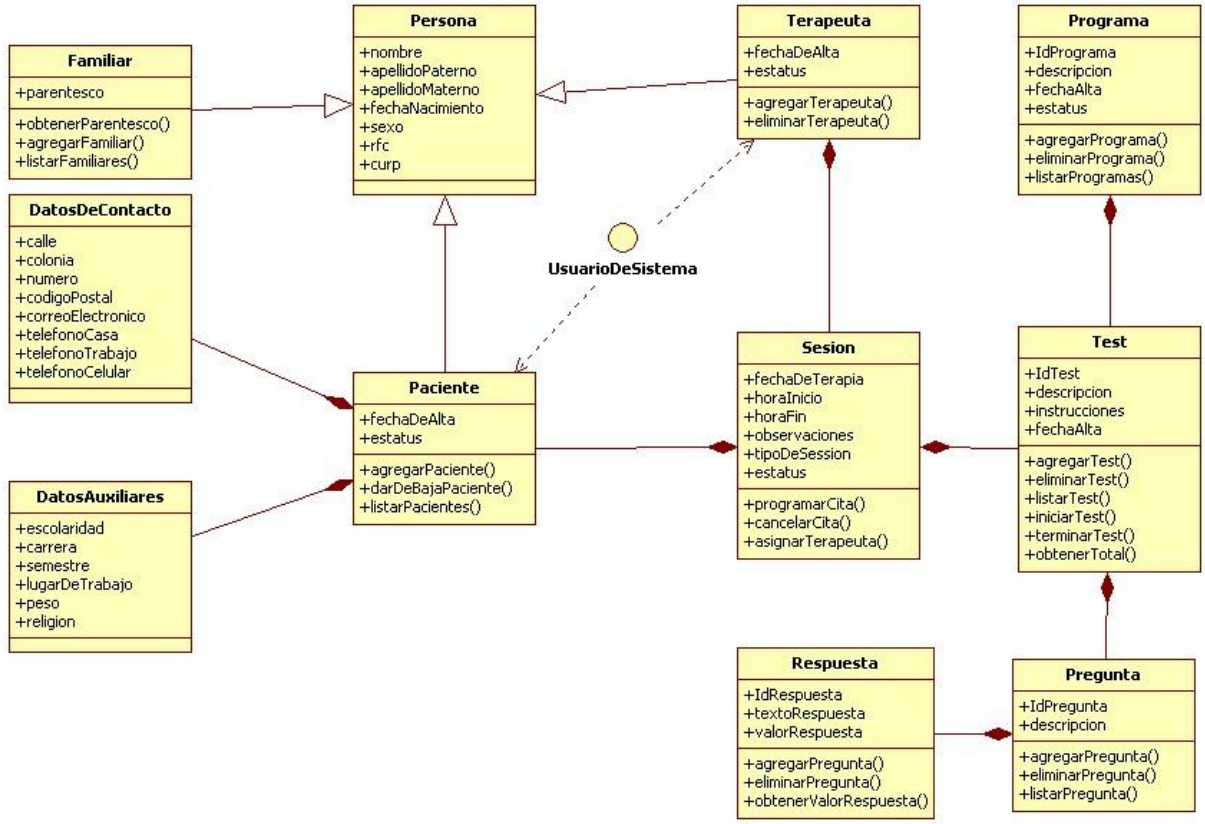


Figura 4.3 Diagrama de clases

En este diagrama podemos ver que la clase Paciente hereda de la clase Persona y a su vez está compuesta por la clase DatosDeContacto y DatosAuxiliares que permiten guardar toda la información que se necesita de un paciente en el sistema.

Por otro lado tanto la clase Paciente como la clase Terapeuta, implementan la interface UsuarioDeSistema, esto quiere decir que un paciente y un terapeuta (referidos como objetos dentro del sistema) tienen la habilidad de poder iniciar sesión y con ello hacer uso del sistema.

5.5. Diagrama de Secuencia

El diagrama de secuencias muestra la forma en que se relacionan los objetos para satisfacer un caso de uso. Aunque los diagramas de secuencias pueden pasar por alto algunos otros objetos que van a interactuar en el sistema, se debe representar la secuencia lógica general que satisface el requerimiento de uso.

Los diagramas de secuencia se conformar por objetos que se representan de forma visual de la siguiente forma: rectángulos con nombre (subrayado), mensajes representados por líneas

continuas y el tiempo representado por una línea vertical. Como ejemplo de diagrama de secuencia se muestra en la figura 4.4 el diagrama del caso de uso “Dar de alta paciente”.

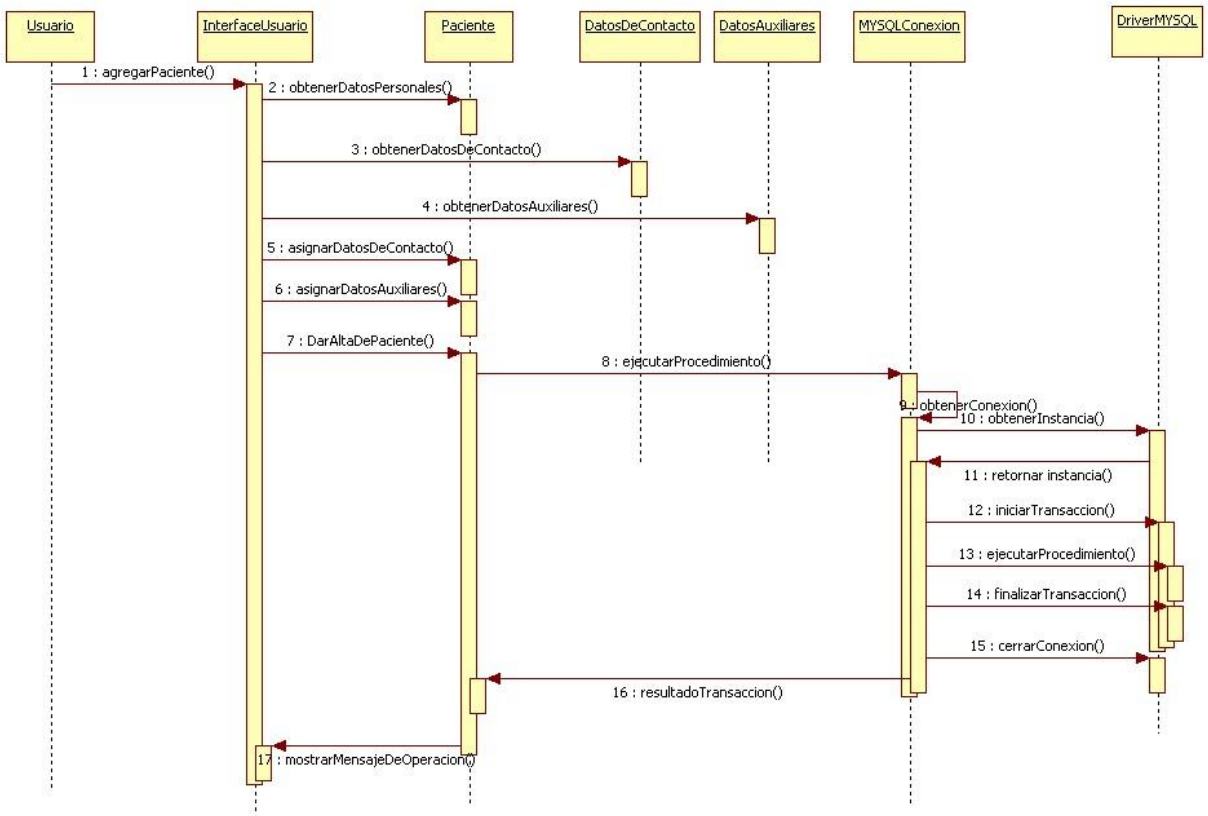


Figura 4.4 Diagrama de secuencia

5.6. Diagrama de Paquetes

Los paquetes es la forma de organizar y agrupar las clases de acuerdo al grupo de objetos que describen. Los paquetes permiten llevar un control y consistencia de las clases utilizadas en el sistema, permitiendo su distribución de una manera más eficiente.

Físicamente, los paquetes son representados por carpetas que van de los mas general a grupos generales de clases. El diagrama de paquetes de la aplicación se muestra a continuación en la figura 4.5

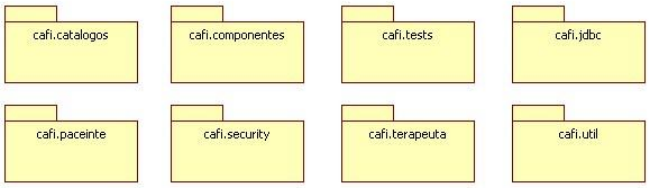


Figura 4.5 Diagrama de paquetes

Como puede observarse, existe un paquete general que agrupa a todos los demás denominado “cafi”, que se refiere al paquete más general de la aplicación y en que están contenidos todos los demás. La descripción de cada paquete se lista a continuación:

- cafi.catalogos : Este paquete almacena las clases de todos los catálogos que se utilizaran en el sistema.
- cafi.componentes: Este paquete almacena todos aquellos componentes que se han creado o bien que fueron sobrescritos para adaptarlos a las necesidades de la aplicación.
- cafi.test: Este paquete almacena las clases que tienen que ver con la realización y mantenimiento de los “Test” que se realizan a los pacientes.
- cafi.jdbc: Esta paquete almacena las clases necesarias para realizar la comunicación con el manejador de la base de datos.
- cafi.paciente: Contiene toda la colección de clases necesarias para realizar las operaciones sobre los pacientes.
- cafi.security: Almacena las clases que tienen que ver con la seguridad de la aplicación.
- cafi.terapeuta: Contiene la colección de clases relacionadas con un terapeuta.
- cafi.util: Contiene clases con características comunes y útiles dentro del sistema.

5.7. Elección de Tecnologías.

De acuerdo a la propuesta de solución, se va a utilizar el Modelo Vista Controlador para implementar el sistema, las tecnologías elegidas dependerán de lo que se tiene disponible y de un análisis de las ventajas y desventajas de utilizar una u otra, enseguida se muestra el diagrama de distribución UML, en la figura 4.6, que representa lo anterior.

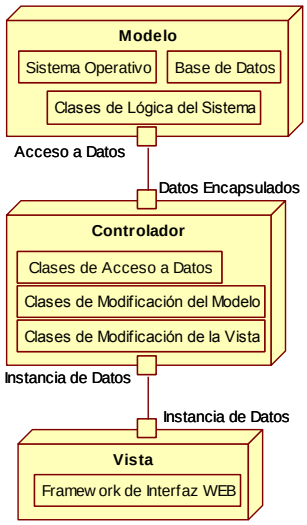


Figura 4.6 Modelo Vista Controlador

La PC en la que se va a instalar el sistema de acuerdo a los recursos con los que cuenta el centro Acasulco, tiene las siguientes características: Windows XP Profesional SP3 instalado en una PC con procesador Intel Pentium IV a 2.4 GHz y 1 GB de RAM a 533 MHz. Por lo tanto, las tecnologías restantes a seleccionar son las siguientes:

- Base de Datos (Oracle 10g Express Edition, SQL Server 2005 Express o Mysql 5.0 Basic).
- Lenguaje de Programación que pueda implementar MVC (Java, ASP .NET o PHP).
- Framework de Interfaz de usuario (QCodo (PHP), Spring .NET (ASP .NET), Struts (Java) o ZK (Java)).

5.8. Selección del Manejador de Base de Datos.

La selección del manejador de base de datos se va a realizar a partir de 3 opciones: Oracle 10g Express Edition, SQL Server 2005 Express y Mysql 5.0 Enterprice, estas ediciones se eligieron ya que no tienen ningún costo de licencia.

La primera comparación es el uso de recursos para la instalación de cada manejador de base de datos:

Base de Datos	Procesador			Memoria RAM		Disco Duro
	Mínimo	Recomendado	Sistema Operativo	Mínimo	Recomendado	
SQL Server 2005 Express	Pentium III a 600 MHz	Más rápido que Pentium III a 1 GHz	Windows 2000 Server SP4, XP SP, Server 2003 y Small Business Server 2003 SP1	512 MB	1 GB	500 MB para instalación y el resto para datos
Oracle 10g Express Edition	Pentium III a 550 MHz	Mayor a Pentium III a 1 GHz	Windows 2000 SP1, Server 2003, XP Professional y Vista	256 MB	512 MB	2.5 GB para memoria virtual e instalación y hasta 4 GB para datos
Mysql 5.0 Enterprice	Pentium II a 300 MHz	Mayor a Pentium II a 800 MHz	Windows 9x, Me, NT, 2000, XP, y 2003	128 MB	512 MB	200 MB para instalación y el resto para datos

Tabla 4.2 Comparación entre manejadores de base de datos

Con la tabla comparativa anterior obtenida de la página web informativa de cada uno de los distribuidores, se puede notar que el manejador de base de datos que utiliza menos recursos es Mysql 5.0 Enterprice.

Existen aspectos negativos en cuanto a las versiones Express de Oracle y SQL Server 2005 que no suceden en Mysql 5.0 Enterprice. Por ejemplo, para la edición Express de Oracle, se tienen las siguientes limitantes: Lo máximo que se puede almacenar en la base de datos son 4 GB, si el equipo en el que se desea instalar tiene más de un procesador Oracle Express únicamente utiliza uno para su funcionamiento además el máximo en memoria RAM que utiliza es 1 GB y solo un usuario se puede conectar, igualmente para la versión Express de SQL Server 2005 se tienen los siguientes detalles: 1 procesador como máximo, 1 GB de memoria RAM como límite, el espacio máximo para la base de datos son 4 GB, el número máximo de instancias son 16.

En conclusión, si se desea que el sistema pueda crecer en funcionalidad y almacenamiento, se requiere una base de datos que lo soporte, y por lo anterior las versiones Express de SQL Server 2005 y Oracle 10g no lo permiten, por lo tanto se elige a Mysql 5.0 Enteprice como manejador de la base de datos.

5.9. Selección del Lenguaje de Programación.

Para la implementación del Modelo Vista Controlador, se va a seleccionar uno de los siguientes lenguajes de programación: ASP .NET, Java o PHP. Existen muchos otros, sin embargo estos tres son los más populares y donde existe mayor documentación.

Uno de los aspectos para tomar esta decisión, es que se eligió la programación orientada a objetos y es necesario que el lenguaje de programación lo soporte, además que el código de funcionalidad no se mezcle con el de interfaz, es decir que se debe evitar el uso de scriplets. Para ASP .NET, no se tiene problema ya que cumple con lo anterior, Java igualmente cumple con estos requisitos y finalmente PHP también cumple, sin embargo la manera en la que se implementa la orientación a objetos (generación de archivos que se interpretan como si fueran clases) y la falta de documentación para realizarlo (la documentación se dirige en su mayoría a PHP como lenguaje estructurado) hace que se tome la decisión de no considerar a PHP para la implementación del sistema.

Finalmente y aunque los lenguajes ASP .NET y Java son muy similares en características, se llegó a la conclusión de seleccionar a Java como lenguaje de programación, debido a que se tiene mayor experiencia que en ASP .NET, además de que para que ASP .NET funcione, se debe utilizar el servidor WEB IIS (Internet Information Server) de Microsoft para el cual se requiere de una licencia.

5.10. Selección del Framework de Interfaz de Usuario.

Como se pudo notar al inicio del tema, el Framework de Interfaz de Usuario para implementar el Modelo Vista Controlador, depende del lenguaje de programación seleccionado, y al elegir Java nos quedan como opciones los frameworks: Struts y ZK.

Struts tiene mayor tiempo como framework para implementar el Modelo Vista Controlador que ZK y su implementación es simple ya que hay bastante documentación al respecto. A grandes rasgos se trata de configurar el servidor WEB el cual normalmente es Apache Tomcat y de generar clases que sirvan como controlador, y finalmente hacer uso de librerías y etiquetas XML las cuales se compilan e interpretan para generar la Vista. ZK tiene una filosofía de funcionamiento parecida, sin embargo tiene la ventaja de implementar AJAX (Asynchronous JavaScript And XML) al mismo tiempo y de manera sencilla, esto permite hacer aplicaciones WEB llamadas RIA (Rich Internet Aplications). Las ventajas de una aplicación RIA sobre una aplicación WEB normal, es que no es necesario recargar toda la página para intercambiar datos aumentando la interactividad, velocidad y usabilidad de las aplicaciones. En Struts existe la forma de implementar AJAX, sin embargo es bastante complicado y se necesita de mucho código e incluso combinación de Java y JavaScript.

ZK presenta una interfaz nativa más amigable que Struts, además de que puede ser modificada de manera sencilla. Un aspecto que vale la pena mencionar, es que cuando se mostraron demos al usuario, este prefirió el aspecto que bridaba ZK, e hizo énfasis en que la velocidad de respuesta era más aceptable que la de la interfaz realizada con Struts, por lo anterior se selecciona a ZK como Framework de Interfaz al Usuario. Este Framework al igual que Struts cuenta con licencia GPL (General Public License) por lo tanto no genera ningún costo por utilizarlo.

6. DESARROLLO.

6.1.Introducción.

La etapa de desarrollo en el proceso de ingeniería de software tiene la finalidad de tomar los resultados del análisis y diseño del software para transformarlos en instrucciones que realice la computadora. Este proceso se realiza generalmente con uno o más lenguajes de programación que ayuden a satisfacer los requerimientos planteados en un inicio.

6.2.Tecnología Java.

Java es toda una tecnología orientada al desarrollo de software con el cual podemos realizar cualquier tipo de programa. Hoy en día, la tecnología Java ha cobrado mucha importancia en el ámbito de Internet gracias a su plataforma J2EE. Pero Java no se queda ahí, ya que en la industria para dispositivos móviles también hay una gran acogida para este lenguaje.

La tecnología Java está compuesta básicamente por 2 elementos: el lenguaje Java y su plataforma. Con plataforma nos referimos a la máquina virtual de Java (Java Virtual Machine).

6.2.1. Lenguaje Java.

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos.

El lenguaje Java es un lenguaje de programación orientado a objetos, de alto nivel, que permite escribir programas para ser ejecutados en la plataforma Java.

6.2.2. Plataforma Java.

La plataforma Java es el nombre de un ambiente o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común.

La plataforma Java se compone de:

1. La maquina Virtual de Java (Java Virtual Machine, JVM)
   - Interprete
   - Conjunto de APIS(Aplication Programable Interface)

2. Plataforma de desarrollo.
   - Compilador
   - Depurados
   - Generados de documentación.
   - Compresor de archivos.

La plataforma Java se divide de cuerdo al tipo de aplicaciones de la siguiente manera:

- Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o Java SE (antes J2SE)
- Plataforma Java, Edición Empresa (Java Platform, Enterprise Edition), o Java EE (antes J2EE)
- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o Java ME (antes J2ME)

Todas las implementaciones de la JVM incluyen la implementación de Java Runtime Enviroment (JRE), la cual se encarga de interpretar un archivo llamado bytecode que es generado por el compilador y que se puede ejecutar en cualquier JVM independiente del sistema operativo o plataforma de hardware.

El proceso de ejecución de un archivo java se muestra en la figura 5.1

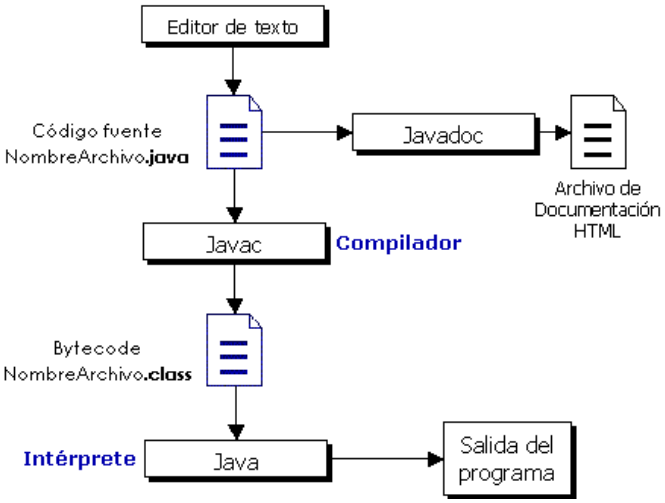


Figura 5.1 Proceso de ejecución de un programa en java

6.3.Conceptos Básicos.

6.3.1. Tipos de datos.

En el lenguaje de programación Java, a pesar de ser un lenguaje orientado a objetos, no todo es un objeto. Hay un grupo de tipos que tiene un tratamiento especial: se trata de los tipos “primitivos”, que se usaran frecuentemente en los programas. La razón para el tratamiento especial es que crear un objeto para variables pequeñas y simples no es eficiente, para estos tipos Java vuelve al

enfoque de C y C++. Es decir, en vez de crear un objeto, se crea una variable "automática" que no es una referencia. La variable guarda el valor, y se coloca en la pila para que sea más eficiente.

Java determina el tamaño de cada tipo primitivo. Estos tamaños no varían de una plataforma a otra como ocurre en la mayoría de los lenguajes. Existen 8 tipos de datos primitivos que se listan en la tabla de la 5.1

Tipo de Dato	Tamaño
boolean	verdadero/falso
byte	1 byte
char	2 bytes
short	2 bytes
Int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes

Tabla 5.1 Tipos de datos en Java

6.3.2. Arreglos.

Los arreglos son una secuencia de datos del mismo tipo. Cada elemento del arreglo esta identificado por un índice. Java considera a cada arreglo como un objeto sin importar el tipo de dato que representa al objeto.

```
1    int[] mesas;
2    mesas= new int[3];
```

La línea 1 declara un arreglo de enteros referenciados por la palabra mesa. La línea 2 inicializa la referencia mesas construyendo un arreglo de 3 elementos.

6.3.3. Control de Flujo.

El control del flujo es la manera que tiene un lenguaje de programación de provocar que el flujo de la ejecución avance y se ramifique en función de los cambios de estado de los datos. La ramificación, iteración, selección y llamadas a subrutina son formas de control de flujo.

- if – else: La construcción if-else provoca que la ejecución atraviese un conjunto de estados boolean que determinan que se ejecuten distintos fragmentos de código.

  if (expresión-booleana) sentencia1; [else sentencia2;]

- switch: La sentencia switch proporciona una forma limpia de dirigir la ejecución a partes diferentes del código en base al valor de una variable o expresión. Esta es la forma general de la sentencia switch:

```
switch ( expresión ) {
        case valor1:
                break;
        case valor2:
                break;
        case valorN:
                break;
        default:
}
```

- while: Ejecuta una sentencia repetidamente mientras una expresión booleana sea verdadera. Esta es su forma general:

```
while ( terminación ) {
   cuerpo;
   [ iteración; ]
}
```

- do-while: La construcción do-while se utiliza cuando se desea ejecutar el cuerpo de un bucle while al menos una vez, incluso si la expresión booleana tiene el valor false la primera vez. Es decir si se desea evaluar la expresión de terminación al final del bucle en vez de al principio como en el while.

```
do { cuerpo; [ iteración; ] } while ( terminación );
```

- for : La sentencia for es una forma compacta de expresar un bucle.

```
for ( inicialización; terminación; iteración ) cuerpo;
```

6.3.4. Referencias y Objetos.

Las referencias son declaraciones de tipo de datos que nos permiten a acceder al espacio de memoria donde reside un objeto. Los objetos se crean haciendo uso de la palabra new y se acceden mediante la referencia asignada.

```
1    Alumno alumno1;
2    alumno1= new Alumno();
```

En la línea 1 se declara una referencia de tipo Alumno llamada alumno1, en este momento la referencia no apunta ningún objeto, es decir apunta a un objeto null (nulo), la línea 2 crea un nuevo objeto de tipo Alumno y asigna la referencia alumno1 a ese objeto.

6.3.5. Excepciones y manejo de errores

La característica de manejo de errores se ha vuelto un punto indispensable en los lenguajes de programación. Java diferencia dos eventos comunes que pueden provocar un flujo distinto en la ejecución de un programa:

Errores: Un error ocurre normalmente de manera interna en la JVM y es poco manejable para el programador de un sistema.

Excepción: Son un tipo de error que se puede controlar en el flujo de ejecución de un sistema en caso de existir.

Un fragmento de código que puede generar error es encerrado en un bloque especial para su tratamiento.

```
try {
//código que puede generar Error
}catch( Exception 1) { //tratamiento en caso de excepción 1}
catch(Exception 2) {//tratamiento en caso de excepción 2}
finally(){ //bloque final, se ejecuta siempre ocurra o no una excepción}
```

6.3.6. Recolector de Basura.

Los programadores conocen la importancia de la inicialización, pero a menudo se les olvida la importancia de la limpieza. Java tiene un recolector de basura para recuperar la memoria de los objetos que ya no se usan, es decir de los cuales se ha perdido su referencia.

El recolector de basura se ejecuta en un proceso en segundo plano cada determinado tiempo, verifica que objetos han dejado de usarse y libera esa porción de memoria al sistema operativo para su nueva reasignación.

6.3.7. Documentación

Hay dos tipos de comentarios en Java. El primero es el estilo de comentarios tradicional de C, que fue heredado por C++. Estos comentarios empiezan por /* y pueden extenderse a lo largo de varias líneas hasta encontrar */.

```
/* ejemplo de comentario */
```

Una de las partes más interesantes del lenguaje Java es que los diseñadores no sólo tuvieron en cuenta que la escritura de código era la única actividad importante, sino que también pensaron en la documentación del código. Esto se hizo mediante comentarios especiales que se incrustan dentro del código fuente, sin embargo, es necesaria una sintaxis especial y una herramienta para extraer esos comentarios.

```
/**
* ejemplo de comentario para javadoc.
*/
```

La herramienta para extraer comentarios se le denomina javadoc. Utiliza parte de la tecnología del compilador Java para extraer etiquetas de comentarios especiales. La salida de javadoc es un archivo HTML que puede visualizarse a través del navegador Web. Gracias a javadoc se tiene

incluso un estándar para la creación de documentación, tan sencillo que se puede incluso esperar o solicitar documentación con todas las bibliotecas Java.

6.3.8. JDBC

Los programas Java con conexión a BD utilizan un driver para conectarse. Un driver es un conjunto de clases que implementan la especificación JDBC para un manejador de base de datos especifico.

6.4.Framework de Desarrollo.

A medida que la Web se ha convertido en la plataforma para crear por defecto desarrollo de aplicaciones, esta arquitectura enfrenta un importante desafío: la incapacidad para representar visualmente la complejidad de las aplicaciones de hoy. Por ejemplo, para dar vender un producto a un cliente, podríamos tener que abrir una página para buscar el registro del cliente, otra para verificar los precios y otra para registrar la venta del producto. De esta forma, los usuarios se ven obligados a abandonar la página en la que se está trabajando y navegar entre varias, de esta manera es fácil perderse y confundirse.

6.4.1. Aplicaciones con Ajax.

Las aplicaciones Web nacieron como contenido HTML estático evolucionaron a HTML dinámico, Applets, Flash y finalmente a la tecnología Ajax (Asynchronous JavaScript and XML). Ajax permite crear aplicaciones con el mismo nivel de interactividad y respuesta como las aplicaciones de escritorio. A diferencia de los Applets o Flash, Ajax se basa en los estándares de los navegadores y JavaScript, además de no requerir ningún plugin adicional.

Ajax es un nuevo tipo de creación de HTML dinámico. Trabaja sobre JavaScript para registrar eventos y sucesos de la actividad del usuario, y con ello realizar una manipulación visual de los resultados en el navegador dinámicamente. La figura 5.2 muestra las diferencias entre una aplicación Web tradicional (Izquierda) y una aplicación con Ajax (Derecha).

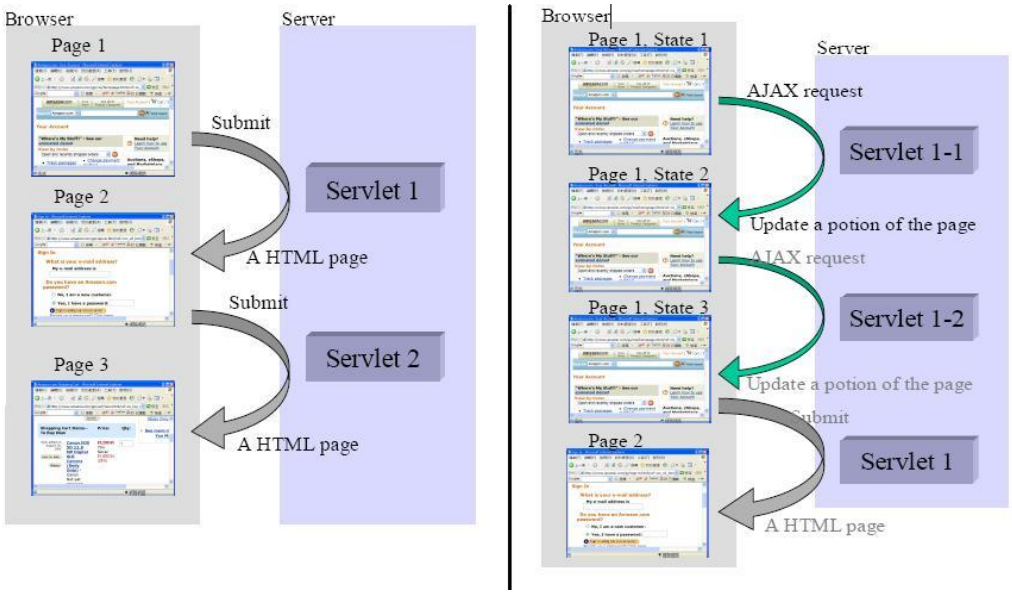


Figura 5.2 Aplicación Web y Ajax

El framework ZK es un manejador de eventos, permite crear agradables e interactivas interfaces de usuario. ZK provee un motor manejador de eventos basado en Ajax, un variado set de XUL (XML para interfaces de usuario) y XHTML (Lenguaje Extensible de Marcado de Hipertexto), además de un lenguaje de marcado propio llamado ZUML (ZK User Interface Markup Language).

6.4.2. Arquitectura del Framework.

ZK incluye un mecanismo basado en Ajax para la interactividad automática, un variado set de componentes basados en XUL y un lenguaje de marcado para simplificar el desarrollo.

El mecanismo basado en Ajax consiste de tres partes: cargador ZK (ZK loader), Motor ZK AU (ZK AU Engine) y un motor de cliente ZK (ZK client Engine). La figura 5.3 muestra la interacción de estos tres componentes.

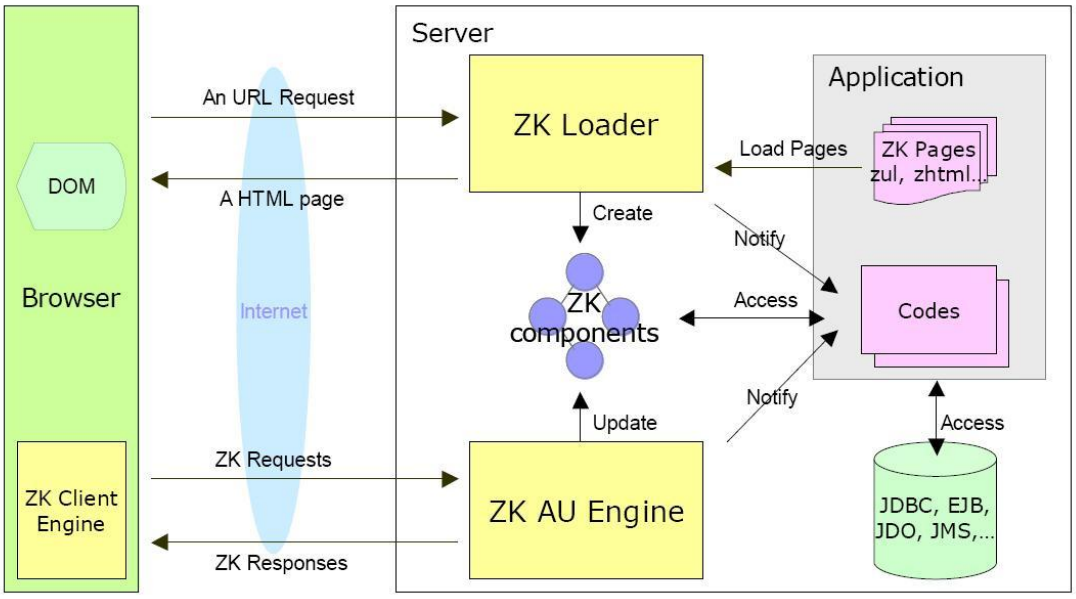


Figura 5.3 ZK loaderm ZK AU Engine, ZK Client Engine

Basados en la petición del usuario, el cargador ZK carga una página ZK, la interpreta y dibuja el resultado como una página HTML como respuesta al URL solicitado. Una página ZK se escribe en un lenguaje de marcado llamado ZUML. ZUML como HTML, se usa para describir que componentes son creados y como se representan visualmente. Estos componentes una vez creados, están disponibles hasta que la sesión expira.

El motor de ZK AU (ZK AU Engine) y el cliente ZK (ZK Client) trabajan juntos lanzando y atrapando. Ellos entregan los eventos que ocurren en el navegador al servidor, actualizan y redibujan los componentes de la aplicación. Este modelo es llamado manejador de eventos.

6.4.3. Flujo de Ejecución.

1. Cuando un usuario solicita una URL, la petición es enviada al servidor Web. El cargador ZK es invocado para atender la solicitud.
2. El cargador ZK carga la pagina especificada y la interpreta para crear apropiadamente los componentes.
3. Después de interpretar toda la pagina, el cargador dibuja el resultado dentro de paginas HTML. La pagina HTML es enviada de vuelta al navegador o browser acompañada del motor cliente ZK (ZK Client Engine).
4. El motor Cliente ZK espera a detectar cualquier evento lanzado por la actividad del usuario, como mover el mouse o cambiar un valor. Cuando lo detecta lo notifica al motor ZK AU (ZK AU Engine) enviando una solicitud ZK (una solicitud mediante Ajax comúnmente llamada Ajax Request).

5. El motor ZK AU (ZK AU Engine) recibe la solicitud del ZK Cliente Engine, AU Engine actualiza el contenido del componente correspondiente si es necesario. Después AU Engine notifica a la demanda de un manejador de eventos si fuera el caso.
6. Si la aplicación selecciona cambiar, agregar o mover componentes, AU Engine envía el nuevo contenido o cambio al Cliente Engine usando una respuesta ZK (ZK response).
7. La respuesta ZK (ZK response) es un comando al Client Engine de cómo actualizar la pagina del navegador.

Como ejemplo del lenguaje de marcado ZUL y la ejecución del programa se muestra el típico hola mundo:

```
<window title="Primer Ventana" border="normal" width="200px">Hola Mundo</window>
```

El código anterior ejecutado sobre el framework ZK produce el resultado de la figura 5.4

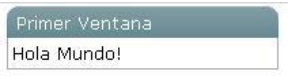


Figura 5.4 Hola Mundo

6.5.Convenciones de codificación.

Al realizar la codificación de un programa no solo resulta importante obtener el resultado esperado, sino seguir normas o estándares que garanticen que un desarrollo pueda ser modificado por otra persona diferente al autor original, con el fin de garantizar un periodo de mantenimiento al terminar el proyecto de software.

La misma especificación del lenguaje Java, propone estándares de codificación tal como se menciona a continuación:

1. Todos los archivos Java deben contener una sola clase o interfase pública.
2. La estructura de un archivo Java es la siguiente:
   - Descripción de la clase y autor de la misma.
   - Sentencia de empaquetamiento: package.
   - Sentencia de importación de clases: import.
   - Declaración de la clase.
3. La declaración de la clase se forma de la siguiente manera:
   - Comentario de documentación de la clase.
   - Sentencia class y nombre de la clase.
   - Variables de clase
   - Variables de instancia.
   - Constructores.
   - Métodos.
4. Las líneas no deben ser de más de 80 caracteres.

5. Cada declaración se hace en una línea.

Convenciones de nomenclatura

1. Paquetes. Inician con el nombre de dominio de primer nivel.
2. Clases e Interfaces. Inician con la primera letra en mayúscula, si el nombre tiene más de una palabra, cada inicio de palabra debe inicial con mayúscula. Las clases denotan regularmente sustantivos y las Interfaces denotan habilidades o comportamientos, ejemplo de nomenclatura: ElefanteRosa
3. Métodos: Inician con letra minúscula, si el nombre del método está compuesto por más de una palabra, la primer palabra ira en minúsculas y se concatenan las palabras con la primer letra de cada una en mayúsculas. Ejemplo vuelaPorElParque()

4.- Variables: Deben tener un nombre significativo y siguen la misma regla que el nombramiento de los métodos.

5.- Constantes: Las constantes van precedidas por la palabra final y deben nombrarse con todas las letras en mayúsculas, en caso de contener más de una palabra se separan por el guio bajo '_';

6.6.Arquitectura del sistema.

Teniendo en cuenta el principio de modularidad, proponemos crear un sistema que trabaje bajo un esquema cliente servidor de 3 capas sobre la plataforma Web.

La interfaz gráfica se mostrará a través del navegador Web, cada petición o evento realizado por el usuario pasa por la arquitectura de ZK para posteriormente hacer uso de las clases que definen el comportamiento de acuerdo a la lógica del negocio. La lógica del negocio interactúa con la aplicación y con la base de datos. La interacción con la base de datos se realiza a través de una interfase de conexión propia de la aplicación y que hace uso del driver de conexión para comunicarse con la base de datos a través de procedimientos almacenados. La arquitectura se muestra en el diagrama de la figura 5.5

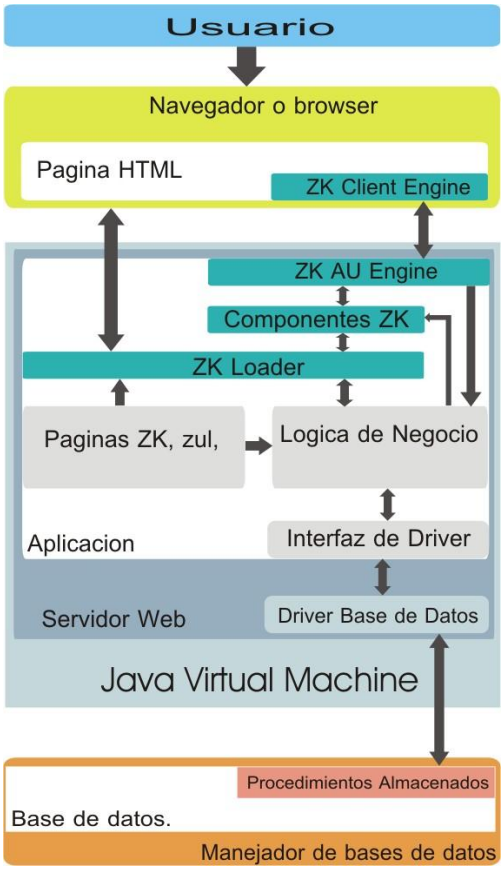


Figura 5.5 Arquitectura del sistema

6.7.Codificación y Documentación.

La etapa de programación se conoce comúnmente como la etapa de programación; que consiste en llevar a código fuente en el lenguaje de programación elegido, todo lo diseñado en las fases anteriores.

Es común pensar que la etapa de programación o codificación, también llamada implementación, es la que insume la mayor parte del trabajo de desarrollo del software; sin embargo, esto puede ser relativo ya que las etapas previas son cruciales, críticas y pueden llevar bastante más tiempo. Se suele hacer estimaciones de un 30% del tiempo total consumido en la programación, pero esta cifra no es consistente ya que depende en gran medida de las características del sistema, su criticidad y el lenguaje de programación elegido.

La implementación del sistema se facilita cuando tenemos un patrón de codificación definido, por lo que toda la programación se realiza de manera similar. A manera de ejemplo se muestra el código fuente de la gestión del catalogo de “Tests” en el sistema.

Los archivos involucrados son:

- Pruebas.zul: Provee de una interfaz al usuario con las opciones que se pueden realizar sobre el catalogo de “Test”.
- Script de interacción: Es el script que atiende los diferentes eventos que realiza el usuario.
- Prueba.java: Contiene la lógica que debe ejecutar cada operación solicitada por el usuario devolviendo el resultado de las misma.
- ConexionMYSQL.java: Es una clase que provee formas simples y genéricas de ejecutar operaciones sobre el driver, que a su vez las enviará a la base de datos.

Pruebas.zul

```
<groupbox id="gb" mold="3d" width="845px">
<caption label="Selecciona una operacion a realizar:"/>
    <hbox>
        <radiogroup id="opcion">
            <radio label="Agregar" value="add" selected="true"
                    tooltiptext="Agrega una nueva prueba"/>
            <radio label="Listar" value="list"
                    tooltiptext="Muestra un listado con las pruebas dadas de alta"/>
            <radio label="Modificar" value="edit"
                    tooltiptext="Modifica los datos de una prueba existente"/>
            <radio label="Eliminar" value="del" tooltiptext="Elimina una prueba existente"/>
        </radiogroup>
        <button label="Aceptar" onClick="showOpcion()" />
    </hbox>
</groupbox>

<tabbox width="845px" mold="accordion"  id="actionPanel" visible="false">
  <tabs>
        <tab label="Catalogo de Pruebas"/>
  </tabs>
  <tabpanels>
        <tabpanel>
            <vbox id="formDatos" visible="false">
                <label id="labelDescripcion" value="Ingresa los datos de la nueva prueba"/>
                <hbox id="buscarClave" visible="false">
                    <label id="labelBuscar" value="Clave de la prueba que deseas modificar"/>
                    <bandbox id="clavePrueba" constraint="no empty: Escibe la clave de la prueba"
                              tooltiptext="Clave del prueba que deseas modificar">
                        <bandpopup>
                            <vbox>
                            <hbox>
                                <textbox id="claveBuscar" cols="5" maxlength="10"/>
                                <button label="Buscar" onClick="buscar()"/>
                            </hbox>
                            <listbox mold="paging" id="editResultado" pageSize="5" width="200px"
                                    onSelect="clavePrueba.value=self.selectedItem.label;
                                    clavePrueba.closeDropdown();">
                                <listhead sizable="true">
                                    <listheader label="Clave" sort="auto" width="20%"/>
                                    <listheader label="Descripcion" sort="auto" width="80%"/>
                                </listhead>
```

```
                    </listbox>
                </vbox>
            </bandpopup>
            </bandbox>
            <button label="Aceptar" onClick="obtenerDatosPrueba()" />
        </hbox>
        <grid width="830px" visible="false" id="gridForm">
            <columns sizable="true">
                <column Width="150px"/>
                <column />
            </columns>
            <rows>
                <row id="rowClave">
                    <label value="Clave de la Prueba: "/>
                        <textbox id="id_prueba" cols="10" maxlength="10" readonly="true"/>
                </row>
                <row>
                    <label value="Nombre de la Prueba: "/>
                    <textbox id="pru_dscrpcion" cols="50" maxlength="50"
                            constraint="no empty: Escribe el nombre de la prueba"/>
                </row>
                <row>
                    <label value="Instrucciones Generales: "/>
                    <textbox id="pru_instrcciones" cols="70"  rows="4" multiline="true"
                        constraint="no empty: Escibe las instrucciones"/>
                </row>
                <row id="rowAgregar" align="right" visible="false">
                    <label value=" "/>
                    <hbox>
                        <button label="Agregar prueba" onClick="agregarPrueba()"/>
                        <button label="Cancelar" onClick="actionPanel.visible=false;"/>
                    </hbox>
                </row>
                <row id="rowEliminar" align="right" visible="false">
                    <label value=" "/>
                    <hbox>
                        <button label="Eliminar prueba" onClick="eliminarPrueba()"/>
                        <button label="Cancelar" onClick="actionPanel.visible=false;"/>
                    </hbox>
                </row>
                <row id="rowModificar" align="right" visible="false">
                    <label value=" "/>
                    <hbox>
                        <button label="Modificar prueba" onClick="modificarPrueba()"/>
                        <button label="Cancelar" onClick="actionPanel.visible=false;"/>
                    </hbox>
                </row>
            </rows>
        </grid>
    </vbox>
    <vbox id="listPanel" visible="false">
        <hbox>
            <label value="Escribe un filtro de busqueda" />
                <textbox id="filtro" cols="20" maxlength="20" />
                <button label="Listar" onClick="buscarPrueba()"/>
```

```
                </hbox>
                <listbox mold="paging" id="listaResultado" pageSize="5" width="830px">
                    <listhead sizable="true">
                        <listheader label="Clave" sort="auto" width="5%"/>
                        <listheader label="Nombre de la Prueba" sort="auto" width="35%"/>
                        <listheader label="Instrucciones" sort="auto" width="60%"/>
                    </listhead>
                </listbox>
                <button label="Cerrar" onClick="actionPanel.visible=false" width="80px"/>
            </vbox>
        </tabpanel>
    </tabpanels>
</tabbox>
</vbox>
```

Script de Interacción.

```
<zscript>
  import cafi.cuestionario.Prueba;

  protected void showOpcion()
  {
    actionPanel.setVisible(true);
    String operacion=opcion.selectedItem.value;

    formDatos.setVisible(false);
    gridForm.setVisible(false);
    buscarClave.setVisible(false);
    listPanel.setVisible(false);
    iniciar();

    if(operacion.equals("add"))
    {
      formDatos.setVisible(true);
      labelDescripcion.Value="Ingresa los datos de la nueva prueba";
      rowClave.setVisible(false);
      gridForm.setVisible(true);
      rowAgregar.setVisible(true);
    }

    if(operacion.equals("del"))
    {
      gridForm.setVisible(false);
      labelDescripcion.setVisible(false);
      formDatos.setVisible(true);
      labelBuscar.Value="Clave de la prueba que deseas eliminar";
      buscarClave.setVisible(true);
      rowEliminar.setVisible(true);
      pru_dscrpcion.setReadonly(true);
      pru_instrcciones.setReadonly(true);
    }

    if(operacion.equals("edit"))
    {
```

```
        gridForm.setVisible(false);
        labelDescripcion.setVisible(false);
        formDatos.setVisible(true);
        labelBuscar.Value="Clave de la prueba que deseas eliminar";
        buscarClave.setVisible(true);
        rowModificar.setVisible(true);
      }
      if(operacion.equals("list"))
      {
        listPanel.setVisible(true);
      }
}

  protected void iniciar()
  {
    claveBuscar.setRawValue("");
    id_prueba.setRawValue("");
    pru_dscrpcion.setRawValue("");
    pru_instrcciones.setRawValue("");
    rowAgregar.setVisible(false);
    rowEliminar.setVisible(false);
    rowModificar.setVisible(false);
    pru_dscrpcion.setReadonly(false);
    pru_instrcciones.setReadonly(false);
    return;
  }

  protected void agregarPrueba()
  {
    String pru_clve=id_prueba.getValue();
    String descrpcion=pru_dscrpcion.getValue();
    int a=Messagebox.show("Confirmas que deseas agregar este prueba?", "Pregunta", Messagebox.OK |
          Messagebox.CANCEL, Messagebox.QUESTION);
    if (a==Messagebox.OK)
    {
      String instrucciones=pru_instrcciones.getValue();
      Prueba nuevaPrueba=new Prueba(pru_clve,descrpcion,instrucciones);
      int resultado=nuevaPrueba.agregarPrueba();
      if(resultado==1)
      {
        Messagebox.show("La prueba se agregó correctamente", "Aviso", Messagebox.OK, Messagebox.EXCLAMATION);
      }
      else
       {
        Messagebox.show("Ocurrio un error al agregar la prueba, intentalo nuevamente", "Error", Messagebox.OK,
                Messagebox.ERROR);
      }
    }
  }

  protected void buscar()
  {
    String cadena=claveBuscar.getValue();
    Prueba.buscarPrueba(cadena,editResultado,'A');
  }
```

```
protected void obtenerDatosPrueba()
{
   String pru_clve=clavePrueba.getValue();
   Prueba prueba=Prueba.obtenerPrueba(pru_clve);
   if(prueba!=null)
   {
      gridForm.setVisible(true);
      rowClave.setVisible(true);
      id_prueba.Value=prueba.getId_prueba();
      pru_dscrpcion.Value=prueba.getPru_dscrpcion();
      pru_instrcciones.Value=prueba.getPru_instrucciones();
   }
}

protected void eliminarPrueba()
{
    String pru_clve=id_prueba.getValue();
    int a=Messagebox.show("Confirmas que deseas eliminar esta prueba?", "Pregunta", Messagebox.OK |
          Messagebox.CANCEL, Messagebox.QUESTION);
    if (a==Messagebox.OK)
    {
        String descrpcion=pru_dscrpcion.getValue();
        String instrucciones=pru_instrcciones.getValue();
        Prueba bajaPrueba=new Prueba(pru_clve,descrpcion,instrucciones);
        int resultado=bajaPrueba.eliminarPrueba();
        if(resultado==1)
        {
           Messagebox.show("La prueba se eliminó correctamente", "Aviso", Messagebox.OK,
                    Messagebox.EXCLAMATION);
        }
        else
        {
           Messagebox.show("Ocurrio un error al eliminar la prueba, intentalo nuevamente", "Error", Messagebox.OK,
                    Messagebox.ERROR);
        }
    }
}
protected void modificarPrueba()
{
   String pru_clve=id_prueba.getValue();
   String descrpcion=pru_dscrpcion.getValue();
   String instrucciones=pru_instrcciones.getValue();
   int a=Messagebox.show("Confirmas que deseas modificar esta prueba?", "Pregunta", Messagebox.OK |
                    Messagebox.CANCEL, Messagebox.QUESTION);
   if (a==Messagebox.OK)
   {
      Prueba modificarPrueba=new Prueba(pru_clve,descrpcion,instrucciones);
      int resultado=modificarPrueba.modificarPrueba();
      if(resultado==1)
      {
         Messagebox.show("La prueba se modificó correctamente", "Aviso", Messagebox.OK, Messagebox.EXCLAMATION);
      }
      else
```

```
        {
          Messagebox.show("Ocurrio un error al modificar la prueba, intentalo nuevamente", "Error", Messagebox.OK,
                  Messagebox.ERROR);
        }
      }
    }

    protected void buscarPrueba()
    {
      String cadena=filtro.getValue();
      Prueba.listarPruebas(cadena,listaResultado);
    }
```

Prueba.java

```
package cafi.cuestionario;
import cafi.jdbc.ConexionMYSQL;
import cafi.util.ItemsUtil;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.zkoss.zul.Listbox;
import org.zkoss.zul.Listcell;
import org.zkoss.zul.Listitem;

/**
 * Esta clase define un objeto de tipo Prueba dentro del sistema. Una prueba
 * es cada uno de los test que se aplican en un programa determinado. Se compone
 * de un Idenficador, una descripcion y las instrucciones que seran mostradas al
 * inicio de cada prueba. Esta clase contiene los metodos necesarios para administrar
 * el catalogo de Pruebas.
 * @author Hugo Camacho, A. Isaac Hernandez
 */
public class Prueba {
    /**
     * Es el Identificador de la prueba dentro del catalogo.
     */
    private String id_prueba;
    /**
     * Es la descripcion o nombre de la Prueba.
     */
    private String pru_dscrpcion;
    /**
     * Es un texto que contiene las instrucciones que se muestran al inicio de
     * cada prueba.
     */
    private String pru_instrucciones;
    /**
     * Constructor unico de la clase.
     * @param id_prueba: Identificador de la prueba en base al catalogo.
     * @param pru_dscrpcion: Es el nombre o descripcion de la prueba.
     * @param pru_instrucciones: Son las instrucciones que se mostraran al inicio
     * de la prueba.
     */
```

```
  public Prueba(String id_prueba,String pru_dscrpcion,String pru_instrucciones)
  {
     this.id_prueba=id_prueba;
     this.pru_dscrpcion=pru_dscrpcion;
     this.pru_instrucciones=pru_instrucciones;
  }
  /**
   * Obtiene el ID de la prueba.
   * @return El Identificador de la prueba en base al catalogo.
   */
  public String getId_prueba() {
     return id_prueba;
  }
  /**
   * Establece el valor del identificador de la prueba.
   * @param id_prueba: Identificador de la prueba en base al catalogo.
   */
  public void setId_prueba(String id_prueba) {
     this.id_prueba = id_prueba;
  }
  /**
   * Obtiene la descripcion o nombre de la prueba.
   * @return La descripcion de la prueba.
   */
  public String getPru_dscrpcion() {
     return pru_dscrpcion;
  }
  /**
   * Establece el texto con el nombre o descripcion de la prueba.
   * @param pru_dscrpcion: Nombre de la prueba
   */
  public void setPru_dscrpcion(String pru_dscrpcion) {
     this.pru_dscrpcion = pru_dscrpcion;
  }
/**
 * Obtiene el texto que representa las instrucciones que son mostradas al iniciar
 * esta prueba.
 * @return String con las instrucciones de la prueba.
 */
  public String getPru_instrucciones() {
     return pru_instrucciones;
  }
  /**
   * establece el texto que representa las instrucciones que son mostradas al
   * iniciar esta prueba.
   * @param pru_instrucciones String con las instrucciones para esta prueba.
   */
  public void setPru_instrucciones(String pru_instrucciones) {
     this.pru_instrucciones = pru_instrucciones;
  }
  /**
   * Realiza una operacion de Update, Insert, Delete sobre el catalogo de
   * pruebas. Este metodo se invoca llamando a cada uno de los correspondientes
   * metodos: agregarPrueba(), modificarPrueba(), borrarPrueba() de esta clase.
```

```
 * @param operacion Operacion a realizar.
 * @return Numero de registros afectados.
 */
private int ejecutarAccion(int operacion)
{
   int respuesta=0;
   ConexionMYSQL con=new ConexionMYSQL();
   try
   {
      String nombreProcedimiento="{call pr_pruebas(?,?,?,?,?)}";
      java.sql.CallableStatement cs=con.preparaProcedimiento(nombreProcedimiento);
      cs.setInt(1,Integer.parseInt(id_prueba));
      cs.setString(2,pru_dscrpcion);
      cs.setString(3,pru_instrucciones);
      cs.setInt(4,operacion);
      cs.setInt(5,5);
      int parametroDeSalida=5;
      respuesta=con.ejecutarProcedimiento(parametroDeSalida);
   }
   catch(SQLException ex)
   {
      ex.printStackTrace();
   }finally
   {
      con.liberaRecursos();
      System.out.println("resultado: "+respuesta);
      return respuesta;
   }
}
/**
 * Obtiene una instancia de un objeto Prueba en base al identificador que
 * se pasa como parametro.
 * @param clave: Identificador de la prueba.
 * @return instancia de un objeto prueba. null si no se encuentra una
 * ocurrencia del Identificador en el catalogo.
 */
public static Prueba obtenerPrueba(String clave)
{
    ConexionMYSQL con = new ConexionMYSQL();
    ResultSet resultados;
    resultados = con.ejecuta("select * from ca_pruebas where id_prueba="+clave);
    Prueba prueba=null;
    try {
     if (resultados.next())
        prueba=new Prueba(resultados.getString(1),resultados.getString(2),resultados.getString(3));
     }catch(SQLException ex)
     {
        ex.printStackTrace();
     }
    finally
    {
       con.liberaRecursos();
       return prueba;
    }
}
```

```
  /**
   * Realiza la operacion de asociar una prueba con un Programa. Cada prueba puede
   * Estar asociada a 1 o mas programas.
   * @param id_programa: identificador del programa.
   * @param id_prueba: identificador de la prueba.
   * @param operacion que se desea realizar: 1 Asignar prueba, 2 Desasignar prueba.
   * @return
   */
  public static int asignarPrueba(String id_programa,String id_prueba,int operacion)
  {
     int respuesta=0;
     ConexionMYSQL con=new ConexionMYSQL();
     try
     {
        String nombreProcedimiento="{call pr_asignarPrueba(?,?,?,?)}";
        java.sql.CallableStatement cs=con.preparaProcedimiento(nombreProcedimiento);
        cs.setString(1,id_programa);
        cs.setInt(2,Integer.parseInt(id_prueba));
        cs.setInt(3,operacion);
        cs.setInt(4,5);
        int parametroDeSalida=4;
        respuesta=con.ejecutarProcedimiento(parametroDeSalida);
     }
     catch(SQLException ex)
     {
        ex.printStackTrace();
     }finally
     {
        con.liberaRecursos();
        return respuesta;
     }
  }
  /**
   * Inserta todas las ocurrencias resultado de una busqueda dentro de la lista que
   * se envia como parametro.
   * @param cadena: Cadena que se desea buscar dentro del catalogo.
   * @param lista: Lista donde se colocaran los resultados de la busqueda.
   */
  public static void listarPruebas(String cadena,Listbox lista)
  {
     ConexionMYSQL con=null;
           Listitem items=null;
           try
           {
                    ItemsUtil.limpiarListBox(lista);
                    String query="select id_prueba,pru_dscrpcion,pru_instrcciones from ca_pruebas where pru_estdo='A' and (
id_prueba like '%"+cadena+"%' "+
              "or pru_dscrpcion like '%"+cadena+"%')";
                    con = new ConexionMYSQL();
                    ResultSet rs=con.ejecuta(query);
                    while(rs.next())
                    {
                             items=new Listitem();
                             items.appendChild(new Listcell(rs.getString(1)));
```

```
                                items.appendChild(new Listcell(rs.getString(2)));
                items.appendChild(new Listcell(rs.getString(3)));
                                lista.appendChild(items);
                        }
                }catch(Exception ex)
                {
                        ex.printStackTrace();
                }
                finally
                {
                        con.liberaRecursos();
                }
        }
        /**
         * Lista todas las pruebas asociadas a un programa.
         * @param id_programa: Identificador del programa en base al catalogo.
         * @param lista: Lista donde se guardaran los resultados de la busqueda.
         */
        public static void listarPruebaPrograma(String id_programa,Listbox lista)
        {
           ConexionMYSQL con=null;
                Listitem items=null;
                try
                {
                        ItemsUtil.limpiarListBox(lista);
                        String query="select PP.id_prueba,pru_dscrpcion,pro_dscrpcion from ca_pruebas_programa PP, ca_pruebas P,
ca_programas PR "+
                  " where PP.id_prueba=P.id_prueba and PR.id_programa=PP.id_programa and PP.id_programa='"+id_programa+"'";
                        con = new ConexionMYSQL();
                        ResultSet rs=con.ejecuta(query);
                        while(rs.next())
                        {
                                items=new Listitem();
                                items.appendChild(new Listcell(rs.getString(1)));
                                items.appendChild(new Listcell(rs.getString(2)));
                                items.appendChild(new Listcell(rs.getString(3)));
                                lista.appendChild(items);
                        }
                }catch(Exception ex)
                {
                        ex.printStackTrace();
                }
                finally
                {
                        con.liberaRecursos();
                }
        }
        /**
         * Realiza una busqueda de Pruebas en el catalogo en base a una cadena de busqueda
         * y un estado.
         * @param cadena: Cadena que se desea buscar en el catalogo.
         * @param lista: Lista donde se guardaran los resultados de la busqueda.
         * @param estado: A Pruebas en estado de Activas, B en estado de Baja.
         */
        public static void buscarPrueba(String cadena,Listbox lista, char estado)
```

```
  {
            ConexionMYSQL con=null;
            Listitem items=null;
            try
            {
                        ItemsUtil.limpiarListBox(lista);
                        String query="select id_prueba,pru_dscrpcion from ca_pruebas where pru_estdo='"+estado+"' and ( id_prueba
like '%"+cadena+"%' "+
                  "or pru_dscrpcion like '%"+cadena+"%')";
                        con = new ConexionMYSQL();
                        ResultSet rs=con.ejecuta(query);
                        while(rs.next())
                        {
                                    items=new Listitem();
                                    items.appendChild(new Listcell(rs.getString(1)));
                                    items.appendChild(new Listcell(rs.getString(2)));
                                    lista.appendChild(items);
                        }
            }catch(Exception ex)
            {
                        ex.printStackTrace();
            }
            finally
            {
                        con.liberaRecursos();
            }
  }
  /**
   * Realiza la operacion de Alta en el catalogo de Pruebas.
   * @return El numero de registros afectados.
   */
  public int agregarPrueba()
  {
    this.id_prueba="0";
    return ejecutarAccion(1);
  }
  /**
   * Realiza la operacion de baja dentro del catalogo.
   * @return numero de registros afectados.
   */
  public int eliminarPrueba()
  {
     return ejecutarAccion(2);
  }
  /**
   * Realiza la operacion de Actualizacion dentro del catalogo.
   * @return numero de registros afectados.
   */
  public int modificarPrueba()
  {
     return ejecutarAccion(3);
  }
  /**
   * Realiza la operacion de Reactivar una prueba dada de baja.
```

```
    * @return numero de registros afectados.
    */
   public int recoverPrueba()
   {
      return ejecutarAccion(4);
   }
}
```

ConexionMYSQL.java

```
package cafi.jdbc;
import java.sql.*;
/**
 * Esta clase permite enviar las instrucciones necesarias al Driver de Conexion
 * de MYSQL para realizar operaciones sobre la base de datos.
 * @author Hugo Camacho. A. Isaac Hernandez.
 */
public class ConexionMYSQL {
   /**
    * El objeto Conexion de la base de datos.
    */
   public Connection con = null;
   /**
    * El ResultSet de resultados de la transaccion.
    */
   public ResultSet rs = null;
   /**
    * Metadatos que se pueden extraer de la Base de Datos.
    */
   public ResultSetMetaData rsmd = null;
   /**
    * Sentencias que sera ejecutada en el manejador de base de datos.
    */
   public Statement stmt = null;
   /**
    * Un Sentencia con parametros para enviarla al manejador de base de datos.
    */
   public PreparedStatement pstmt = null;
   /**
    * Objeto que nos permite ejecutar un procedimiento en la base de datos.
    */
   public CallableStatement cs= null;
   /**
    * Permite inicializar la conexion a la base de datos.
    */
   public ConexionMYSQL()
   {
            try {
                     // Cargamos el puente JDBC=>MySQL
                     Class.forName ("com.mysql.jdbc.Driver");
                     con = DriverManager.getConnection ("jdbc:mysql://localhost/cafi", "root", "xxx" );
                     stmt = con.createStatement();
            }catch( SQLException ex ) {
                     imprimeSQLEx(ex);
            }catch( Exception ex ) {
```

```
                        System.out.println( "Se produjo un error inesperado: "+ex.getMessage() );
            }//fin catch
}
/**
 * Indica al servidor de MYSQL que se ejecutara una transaccion.
 */
public void iniciarTransaccion()
{
   try
   {
      con.setAutoCommit(false);
   }
   catch(SQLException ex)
   {
      System.out.println("error al iniciar la transaccion");
      ex.printStackTrace();
   }
}
/**
 * Indica al servidor de MYSQl el termino de una transaccion.
 * @param error true si se debe realizar un commit, false si se debe realizar un
 * rollback a la transaccion.
 */
public void terminarTransaccion(boolean error)
{
   try {

      if (error) {

         con.commit();
      } else {
          con.rollback();
      }
   } catch (SQLException ex) {
     System.out.println("Ocurrio un error al terminar la transaccion");
     ex.printStackTrace();
   }
}
/**
 * Iniciar la configuracion para la ejecucion de un procedimiento alamacenado.
 * @param nombreProcedimiento
 * @return
 */
public CallableStatement preparaProcedimiento(String nombreProcedimiento)
{
            try
            {
                        cs=con.prepareCall(nombreProcedimiento);
            }catch( SQLException ex ) {
                                    imprimeSQLEx(ex);
                        }catch( Exception ex ) {
                         System.out.println( "Se produjo un error inesperado: "+ex.getMessage() );
                        }
            return cs;
```

```
}
/**
 * Realiza la ejecucion de un procedimiento alamacenado.
 * @param parametroDeSalida Numero de parametro que inidca la salida
 * del procedimiento.
 * @return numero de registros afectados.
 */
public int ejecutarProcedimiento(int parametroDeSalida)
{
	int respuesta=0;
	try
	{
		cs.execute();
		respuesta=cs.getInt(parametroDeSalida);
	}
	catch(SQLException ex)
	{
		imprimeSQLEx(ex);
	}
	catch( Exception ex ) {
		 System.out.println( "Se produjo un error inesperado: "+ex.getMessage() );
		}//fin catch
	return respuesta;

}
/**
 * Permite realizar la ejecucion de sentencias  SELECT En el manejador de base
 * de datos.
 * @param sql Sentencia SQL .
 * @return ResultSet con los resultados obtenidos.
 */
public ResultSet ejecuta(String sql){//
		try {
		rs = stmt.executeQuery(sql);//se le pasa como parametro al statment un executeQuery
		}catch( SQLException ex ) {
			System.out.println( "Error: SQLException" );
			System.out.println ("SQLState: " + ex.getSQLState ());
			System.out.println ("Mensaje:  " + ex.getMessage ());
			System.out.println ("Vendedor:   " + ex.getErrorCode ());
		}//fin catch
		return rs;
	}
```

7. PRUEBAS Y LIBERACION

7.1.Introducción.

La etapa de pruebas en el proceso de construcción de software tiene la finalidad de probar y corregir aquellos errores que pudieran surgir al ejecutar las funciones del software o bien los errores que pueden llegar a surgir al introducir ciertos datos. Las pruebas se dividen en tipos de acuerdo a qué parte del sistema se desea probar, una vez aprobadas todas las pruebas y dado el visto bueno del usuario se procede a la liberación en su primer versión del sistema.

7.2.Pruebas

El proceso de pruebas o "testing" de un sistema se realiza en 3 etapas que permiten identificar y solucionar los errores de una manera eficaz. Las etapas del proceso son:

- Pruebas Unitarias.
- Pruebas de Integración.
- Pruebas de Usuario.

7.2.1. Pruebas Unitarias y JUnit

La primera etapa de pruebas consiste en realizar una serie de pruebas de manera independiente de cada componente o bien de cada sistema. En esta etapa se realizan pruebas con distintos datos y se comprueba el correcto funcionamiento de procedimientos y algoritmos que realiza el sistema, validando en cada prueba el éxito de estos. Esta etapa recibe el nombre de pruebas unitarias ya que no se ve involucrado la comunicación entre sistemas, limitándose únicamente a probar cada componente por separado.

En el desarrollo con la tecnología java, encontramos un marco de trabajo o "framework" desarrollado con el único propósito de realizar pruebas unitarias sobre desarrollos en java, el conjunto de clases que forma este marco de trabajo se denomina JUnit.

JUnit fue creado por Kent Beck y Erich Gamma, este "framework" permite evaluar si el funcionamiento de los métodos de una clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

7.2.2. Pruebas de Integración

Las pruebas integrales o pruebas de Integración se realizan un vez que se han hecho las pruebas unitarias. Las pruebas de integración consisten en verificar que los sistemas que trabajan juntos lo hagan de manera correcta, esto se logra combinando los módulos y probándolos en grupos.

En nuestro desarrollo de software podemos identificar dos sistemas que deben trabajar de manera conjunta, por un lado la interface del usuario también llamado “aplicativo” y por el otro la base de datos que proveerá de información al aplicativo. La conexión se ilustra en la figura 6.1

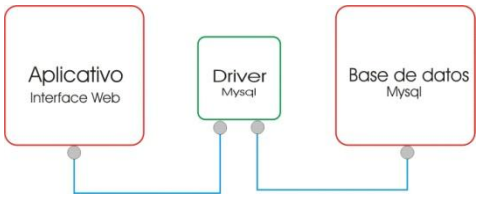


Figura 6.1 Conexiones de sistemas

Dado que el driver de conexión es un componente o subsistema reutilizado que provee el propio desarrollador de la base de datos en este caso MYSQL, podemos definir las pruebas integrales como la pruebas de comunicación y respuesta entre el aplicativo y la base de datos.

7.2.3. Prueba de Usuario

En esta etapa se realizan las pruebas completas del sistema validando en cada caso de prueba que los resultados sean los esperados por parte del equipo de desarrollo y por parte del usuario del sistema.

En esta etapa del desarrollo se mostrará que tan bien se realizó el trabajo en las etapas anteriores, ya que un resultado o funcionalidad no esperada por el usuario puede atrasar en gran medida la liberación del proyecto además de aumentar los costos que se previeron en un inicio.

Como resultado de esta etapa de pruebas, el usuario validará y emitirá vistos buenos que permitirán iniciar la fase de liberación.

7.3. LIBERACIÓN

La liberación es un proceso en el cual el producto terminado pasa de un ambiente de pruebas a un ambiente de producción. La liberación final requiere de un manual de instalación que permita realizar la distribución del sistema.

7.3.1. Requisitos previos.

Dado que el sistema desarrollado es una aplicación Web programada en java, se requiere un servidor de aplicaciones Web que soporte aplicaciones java y la maquina virtual de java que permita interpretarla. Los servidores de aplicación Web más comunes de este tipo son:

- Servidor Apache Tomcat: Es un servidor Web desarrollado sobre la plataforma java su distribución es libre.
- Servidor Apache: Es el servidor Web más utilizado, sin embargo para poder ejecutar aplicaciones java es necesario instalar componentes adicionales.
- WebLogic: Es un servidor Web desarrollado sobre la plataforma java, sin embargo su uso requiere compra de licencias con su actual propietario BEA.
- GlassFish: Servidor Web desarrollado en java por la comunidad de desarrollos de SUN, su distribución es libre.

Cualquier servidor Web de los anteriormente mencionados permite ejecutar la aplicación de manera correcta. Por otro lado La maquina virtual de java (JVM) requerida, debe ser la versión 5 o posterior.

7.3.2. Manejador de base de datos

El manejador de base de datos específico que utiliza la aplicación es MYSQL versión 5 o posterior. Esto es de suma importancia ya que el driver de conexión solo es capaz de establecer una comunicación con un manejador de base de datos de MYSQL.

Cabe destacar que dado que la aplicación se ejecuta e interpreta por la maquina virtual de java, por un servidor de aplicaciones y por un manejador de base de datos, el sistema operativo y arquitectura de hardware en la que se ejecute son independientes al sistema desarrollado, por lo cual su especificación no es necesaria siempre y cuando se tengan los requisitos previos debidamente instalados.

7.3.3. Requerimientos de hardware en el servidor

- 50 MB de espacio en disco para la instalación de la aplicación.
- 80 MB de espacio en disco para la instalación de la maquina virtual de java (JVM)

- Espacio suficiente para el servidor de aplicaciones Web seleccionado.
- 100 MB de espacio en disco para el manejador de base de datos y la base de datos del sistema. (tamaño inicial que aumentará indefinidamente conforme se gestionen datos en el sistema).
- Memoria RAM de 256 MB, 1 GB recomendable.

7.3.4. Requerimientos de software para el cliente.

- Navegador o explorador Web (FireFox, IE versión 7 o posterior, safari, etc…)
- Acrobat Reader 6 o posterior.

7.3.5. Piezas de instalación.

La aplicación se distribuye a través de 2 únicas piezas:

Archivo.War: Los archivos war son la manera en que se distribuyen las aplicaciones Web de java, contiene todas las piezas y librerías necesarias para instalar la aplicación en cualquier servidor de aplicaciones java. Para realizar la instalación basta con realizar un “deploy” de la aplicación. El proceso de deploy se realiza de manera distinta en cada servidor Web y consiste básicamente en extraer las piezas del archivo war para realizar la instalación de la aplicación, proceso que ocurre de manera automática para el instalador.

Archivo.sql: Contiene la estructura y procedimientos almacenados que se requieren en la base de datos, para realizar la instalación basta con ejecutar el script en el manejador de la base de datos.

8. CONCLUSIONES.

Conclusiones por: Apolo Isaac Hernández Ávila.

Actualmente muchas instituciones médicas necesitan herramientas de software que les permitan desarrollar sus actividades más frecuentes con mayor agilidad. En este caso, el sistema desarrollado, lleva parte de la administración del centro de salud Acasulco enfocándose aún más en la aplicación de test psicológicos.

Para desarrollar esta herramienta, lo primero fue entender el requerimiento haciendo algunas entrevistas con el usuario, con ello se detectó que el problema principal era la generación, aplicación y calificación de pruebas psicológicas ya que esto se realizaba de forma manual lo que consumía mucho tiempo, otro de los problemas que se presentaba, era que no se tenía una base de datos central con información de pacientes, ya que todo se administraba en expedientes de papel lo que dificultaba dar un seguimiento completo a los pacientes.

Teniendo claras las necesidades del usuario, se procedió a elegir la tecnología para desarrollar el sistema y así se optó por JAVA; lo cual llevo a utilizar la metodología orientada a objetos (RUP – Rational Unified Process) para las etapas de análisis y desarrollo. Otro aspecto importante es que como el usuario requería de una interfaz WEB, se eligió el Modelo Vista Controlador (MVC) como arquitectura del sistema, siendo esta una de las mejores prácticas en el desarrollo de este tipo de aplicaciones.

Al terminar el análisis, hubo una etapa de investigación sobre el Framework ZK el cual permite desarrollar aplicaciones RIA (Rich Internet Applications) al implementar AJAX (Asynchronous JavaScript And XML), finalmente se entendió como aplicar la arquitectura MVC y se procedió al desarrollo.

Al finalizar cada uno de los módulos del sistema se hacían pruebas unitarias para verificar que no hubiera defectos. Terminando esto, se procedió a mostrar el funcionamiento de la aplicación al usuario quien dio su visto bueno o ayudó a detectar mejoras en cuanto a la presentación o a la lógica implementada.

Finalmente las pruebas con el usuario ayudaron a facilitar la aceptación en cuanto al uso del sistema. Por lo que se puede concluir que el objetivo de proveer de una herramienta de fácil uso que ayude con las actividades de aplicación de tests y administración de pacientes, se logró satisfactoriamente.

Conclusiones por: Hugo Camacho Pérez.

En la actualidad es difícil imaginar un proceso de negocio de cualquier índole que no involucre o esté ligado a un sistema informático. Los sistemas informáticos aportan un gran plus en el manejo de los datos haciendo que puedan ser obtenidos y explotados de manera eficiente en cualquier momento, logrando resultados de manera eficaz, es decir, en menos tiempo y con el mínimo de recursos empleados.

Para que un sistema informático sea operado en cualquier negocio, refiriéndonos a negocio como cualquier actividad que pueda desempeñar un grupo de personas en una organización o empresa, no basta que tenga la capacidad de almacenar y mostrar datos, es necesario que el sistema que se pretende operar maneje las reglas y los procesos del negocio para que de esta forma los usuarios no cambien su manera de operar y solo se optimicen aquellas tareas que hará el sistema.

El manejo de los procesos de negocio de un sistema trae consigo que antes de desarrollar cualquier sistema se conozcan a detalle dichos procesos por parte de las personas encargadas de desarrollarlo en conjunto con la retroalimentación de los usuarios finales. Esto permitirá saber al equipo de desarrolladores que es exactamente lo que el usuario desea obtener así como las recomendaciones de cómo podría mejorarse. Es importante también definir fronteras o limites del sistema, con esto el usuario sabrá que hará y que no hará el sistema para no crear falsas expectativas.

Una vez que se conocen los procesos de negocio se debe diseñar una solución que permita construir el sistema cumpliendo con los requisitos solicitados. El diseño de la solución no es algo trivial, se debe tomar en cuenta los recursos con que cuenta el negocio o que está dispuesto a invertir, con base a ello se diseña una solución que satisfaga las necesidades. No debemos olvidar que implementar un sistema informático debe contribuir a mejorar los procesos y no a generar más problemas, es decir el sistema debe acoplarse a los procesos y no los procesos al sistema.

Cuando se termina la construcción del producto de software es común pensar que la solución está lista para implementarse, sin embargo antes de implementar cualquier solución se debe pasar por la etapa de pruebas, donde el usuario debe certificar cada una de las funciones del sistema, hasta la mas mínima, y de esta forma evitar futuros malentendidos entre los usuarios y el equipo de desarrollo que implementó la solución. La certificación también debe incluir la aceptación de la interfaz que el usuario operará en el sistema, una interfaz difícil de manejar contribuye a un uso ineficiente del sistema, lo que ocasionara que el usuario opte por no utilizar algunas funciones. Una vez certificado el producto sigue una etapa de capacitación y seguimiento a todos los usuarios para posteriormente, ser implementado en el negocio.

Como parte independiente de la funcionalidad y requisitos del sistema, es importante que el equipo de desarrolladores en conjunto con el usuario, definan un proceso alterno de trabajo en caso de que el sistema deje de funcionar por cualquier motivo, esto con el fin de no detener

aquellos procesos identificados como críticos en el negocio. Definir un proceso alterno también debe incluir un respaldo constante de la información así como su correspondiente mecanismo de restauración.

De esta manera, con el sistema construido para el centro Acasulco de la facultad de psicología se pretende que los terapeutas inviertan menos tiempo en la elaboración de documentos y almacenamiento de datos de los paciente que la calificación y elaboración de resultados sea más rápida y que permita a un terapeuta realizar el análisis de la interpretación de resultados en cualquier momento y cuando así lo desee, lo que antes tomaba hasta una semana hoy por medio del sistema de información puede obtenerse en menos de una hora.

También se redujo el consumo de papel que es otro beneficio que traen los sistemas de información electrónicos, la elaboración de documentos e impresión en papel para ser guardados en folders y estos en archiveros con el espacio que esto representa, se reduce a la impresión de resultados que se desean entregar al paciente si este así lo desea.

Con la capacidad del sistema de interactuar no solo con el terapeuta sino el paciente al poder resolver de manera electrónica los reactivos de los test se mejora la imagen y el servicio que el centro ofrece a sus pacientes.

Con todo lo anterior el éxito del funcionamiento de un sistema no se reduce únicamente a la tecnología utilizada o a la parte de la construcción, el éxito de un sistema engloba varios factores que deben tomarse en cuenta y que al no considerarlos tarde o temprano se reflejara en el descontento del usuario.

9. Anexo A: Casos de Uso.

CU 2	Dar baja de Usuario	
Meta y Contexto	Dar de baja lógicamente al Usuario en el sistema	
Precondiciones	Tener disponibles los datos del Usuario al que se le va a dar de baja Que el Administrador este autentificado en el sistema	
Condición de Éxito	Que no se muestre el Usuario en el sistema y que no se pueda autentificar en este.	
Condiciones de Fallo	Que el Usuario siga habilitado en el sistema	
Actor Principal, Actor(es) Secundarios	Administrador Usuario (Terapeuta o Paciente) , Sistema	
Disparador	Se le pide al Administrador que realice la baja a un Usuario	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige al Usuario en el Sistema
	2	El Sistema pide la confirmación del Administrador
	3	El Administrador confirma que el Usuario es el que quiere dar de baja
	4	El Sistema realiza la baja y muestra una confirmación al Administrador
Flujos Alternativos	**Paso**	**Acción Ramificada**
	3	El Administrador confirma que no es el Usuario que quiere dar de baja: 1. Fin del caso de uso
Variaciones	**Descripción**	
	N/A	

CU 3	Modificar Datos de Usuario
Meta y Contexto	Modificar los Datos del Usuario en el sistema
Precondiciones	Tener disponibles los datos del Usuario al que se le van a modificar sus datos Que el Administrador este autentificado en el sistema
Condición de Éxito	Que los datos modificados durante esta tarea se muestren en el sistema
Condiciones de Fallo	Que los datos modificados durante esta tarea no se reflejen en el sistema
Actor Principal,	Administrador

Actor(es) Secundarios	Usuario (Terapeuta o Paciente) ,Sistema	
Disparador	Se le pide al Administrador que realice una modificación a los datos del Usuario	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige al Usuario en el Sistema
	2	El Administrador modifica los datos del Usuario de acuerdo a su perfil (Terapeuta o Paciente)
	3	El Sistema realiza los cambios y muestra una confirmación al Administrador
Flujos Alternativos	**Paso**	**Acción Ramificada**
	N/A	N/A
Variaciones	**Descripción**	
	Si el Usuario elegido es un Paciente, el sistema mostrará el expediente para que pueda ser modificado al igual que los comentarios asociados a cada terapia. Si el Administrador es un Terapeuta sólo podrá elegir como Usuario a cualquiera de sus Pacientes y únicamente podrá modificar el expediente y los comentarios asociados a cada terapia (quedan fuera los datos personales).	

CU 4	Consultar información de Usuario	
Meta y Contexto	Consultar la información del Usuario en el sistema	
Precondiciones	Tener disponibles los datos del Usuario al que se le va a hacer una consulta de sus datos Que el Administrador este autentificado en el sistema	
Condición de Éxito	Que los datos del Usuario se muestren en el sistema	
Condiciones de Fallo	Que los datos del Usuario valido elegido no se muestren en el sistema	
Actor Principal, Actor(es) Secundarios	Administrador Usuario (Terapeuta o Paciente) ,Sistema	
Disparador	Se le pide al Administrador que realice una modificación a los datos del Usuario	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige al Usuario en el Sistema

	2	El Sistema muestra los datos del Usuario de acuerdo a su perfil (Terapeuta o Usuario)
Flujos Alternativos	**Paso**	**Acción Ramificada**
		N/A
Variaciones	**Descripción**	
	Si el Usuario elegido es un Paciente, el sistema mostrará su expediente y las pruebas asociadas que podrán ser consultadas con sus resultados Si el Administrador es un Terapeuta sólo podrá elegir como Usuario a si mismo o a cualquiera de sus Pacientes	

CU 5	Administrar prueba	
Meta y Contexto	En el centro Acasulco se tiene una variedad de pruebas psicológicas estándar, sin embargo estas pueden cambiar con el tiempo en cuanto a su forma de calificar, preguntas, respuestas y el valor de los reactivos, es por ello que en el sistema se podrán dar de alta, modificar y dar de baja pruebas	
Precondiciones	Tener diseñada la prueba que se va a dar de alta en el sistema, o los datos de la prueba que se va a modificar o dar de baja Que el Administrador este autentificado en el sistema	
Condición de Éxito	Que la prueba esté disponible para ser asignada por los Terapeutas a cada uno de sus Pacientes o que la prueba se haya dado de baja lógicamente según sea el caso	
Condiciones de Fallo	Que la prueba no esté disponible en el sistema o que no se haya dado de baja según sea el caso	
Actor Principal, Actor(es) Secundarios	Administrador Terapeuta, Paciente, Sistema	
Disparador	Se le solicita al Administrador cualquiera de las siguientes tareas: Dar de alta una prueba, modificar una prueba o dar de baja una prueba	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige cualquiera de las siguientes opciones: Dar de alta, modificar o dar de baja a la prueba
	2	El Administrador selecciona dar de alta la prueba
	3	El Administrador captura la prueba según las opciones mostradas en el Sistema
	4	El Sistema confirma el alta de la prueba y la pone disponible para que los Terapeutas puedan asignarla a sus Pacientes
Flujos Alternativos	**Paso**	**Acción Ramificada**

	2	El Administrador selecciona modificar prueba: 1. El Administrador elige la prueba a modificar 2. El Administrador modifica las preguntas, respuestas o reactivos correspondientes a la prueba elegida 3. El Sistema confirma la modificación de la prueba y la pone disponible para que los Terapeutas puedan asignarla a sus Pacientes
	2	El Administrador selecciona dar de baja a la prueba: 1. El Administrador elige la prueba para darla de baja 2. El Sistema pide la confirmación de la baja 3. El Administrador confirma la baja de la prueba 4. El Sistema da de baja lógicamente a la prueba y confirma al Administrador
	2.3	El Administrador decide no dar de baja la prueba: 1. Fin del caso de uso
Variaciones	**Descripción**	
	N/A	

CU 6	Programar prueba	
Meta y Contexto	En el centro Acasulco se tiene una variedad de pruebas psicológicas estándar, sin embargo estas pueden cambiar con el tiempo en cuanto a su forma de calificar, preguntas, respuestas y el valor de los reactivos, es por ello que en el sistema se podrán dar de alta, modificar y dar de baja pruebas	
Precondiciones	Tener diseñada la prueba que se va a dar de alta en el sistema, o los datos de la prueba que se va a modificar o dar de baja Que el Administrador este autentificado en el sistema	
Condición de Éxito	Que la prueba esté disponible para ser asignada por los Terapeutas a cada uno de sus Pacientes o que la prueba se haya dado de baja lógicamente según sea el caso	
Condiciones de Fallo	Que la prueba no esté disponible en el sistema o que no se haya dado de baja según sea el caso	
Actor Principal, Actor(es) Secundarios	Administrador Terapeuta, Paciente, Sistema	
Disparador	Se le solicita al Administrador cualquiera de las siguientes tareas: Dar de alta una prueba, modificar una prueba o dar de baja una prueba	
Escenario principal	**Paso**	**Acción**
	1	El Administrador elige cualquiera de las siguientes opciones: Dar de alta, modificar o dar de baja a la prueba

	2	El Administrador selecciona dar de alta la prueba
	3	El Administrador captura la prueba según las opciones mostradas en el Sistema
	4	El Sistema confirma el alta de la prueba y la pone disponible para que los Terapeutas puedan asignarla a sus Pacientes
Flujos Alternativos	**Paso**	**Acción Ramificada**
	2	El Administrador selecciona modificar prueba: 4. El Administrador elige la prueba a modificar 5. El Administrador modifica las preguntas, respuestas o reactivos correspondientes a la prueba elegida 6. El Sistema confirma la modificación de la prueba y la pone disponible para que los Terapeutas puedan asignarla a sus Pacientes
	2	El Administrador selecciona dar de baja a la prueba: 5. El Administrador elige la prueba para darla de baja 6. El Sistema pide la confirmación de la baja 7. El Administrador confirma la baja de la prueba 8. El Sistema da de baja lógicamente a la prueba y confirma al Administrador
	2.3	El Administrador decide no dar de baja la prueba: 2. Fin del caso de uso
Variaciones	**Descripción**	
	N/A	

10. Anexo B: Diagrama de Flujo de Datos.

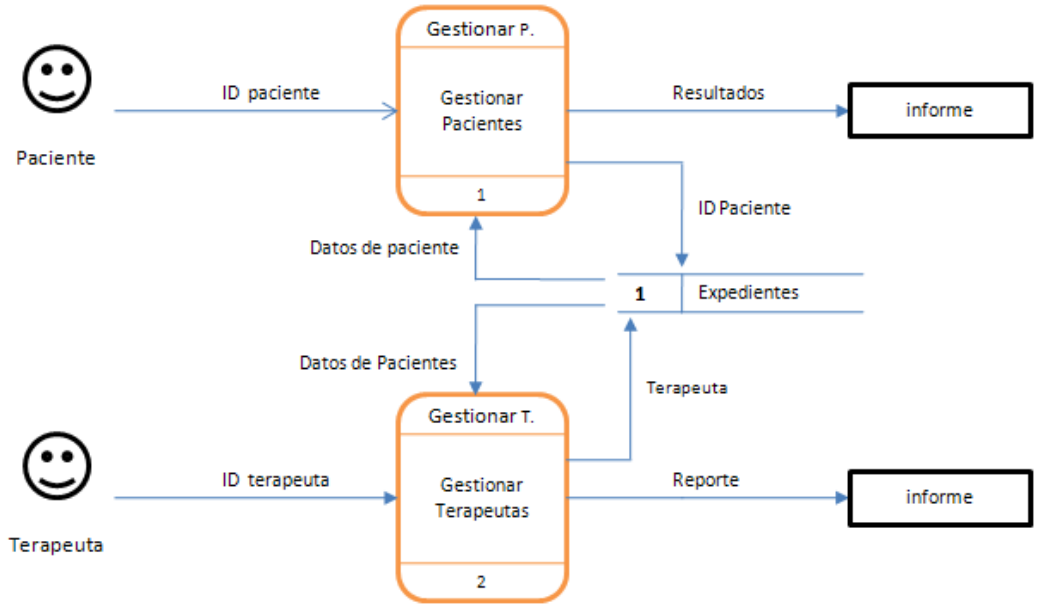


Diagrama de flujo de datos nivel 0.

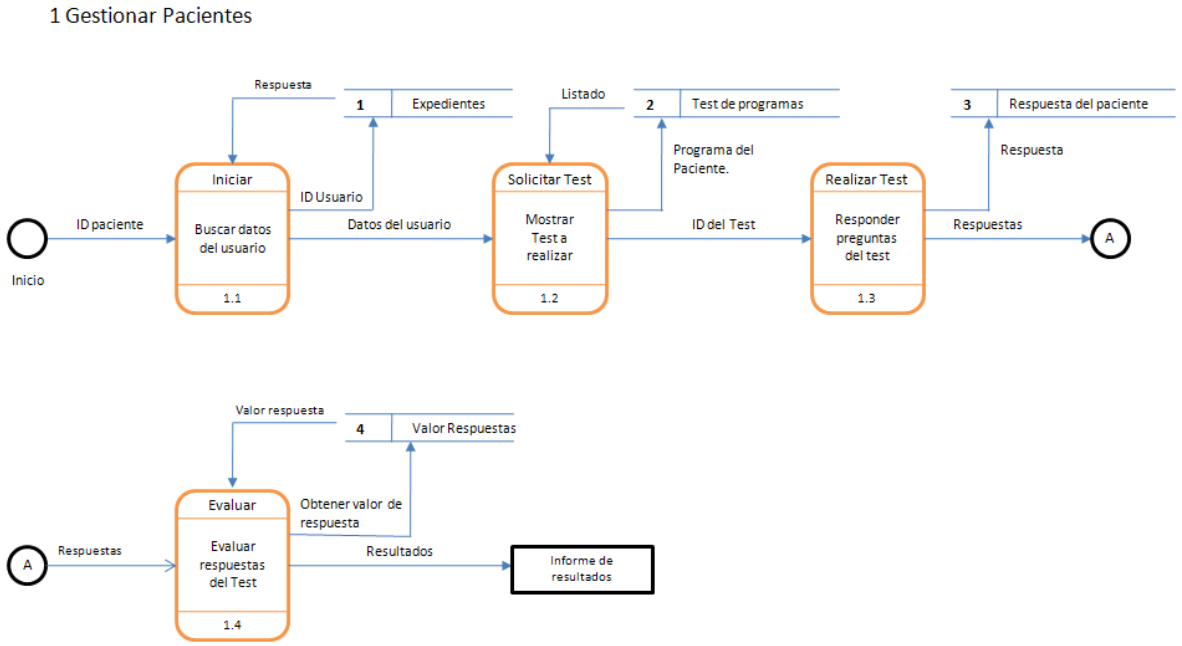


Diagrama de flujo de datos nivel 1.

2 Gestionar Terapeutas

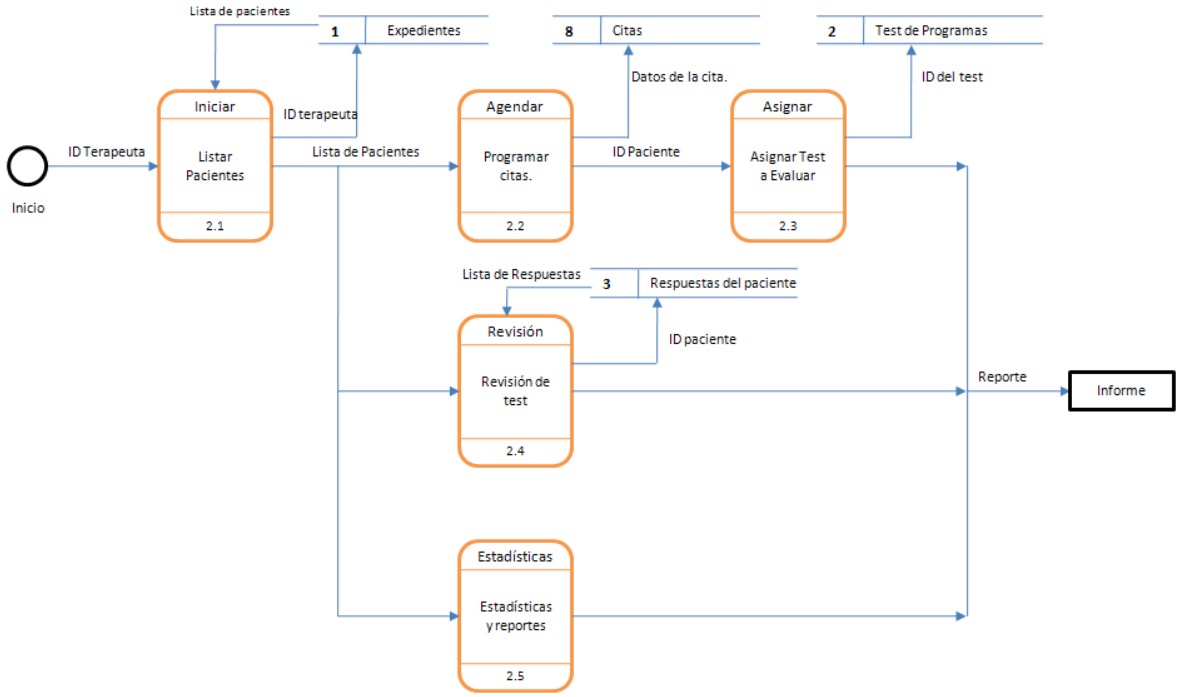


Diagrama de flujo de datos nivel 1.

1.3 Realizar Test

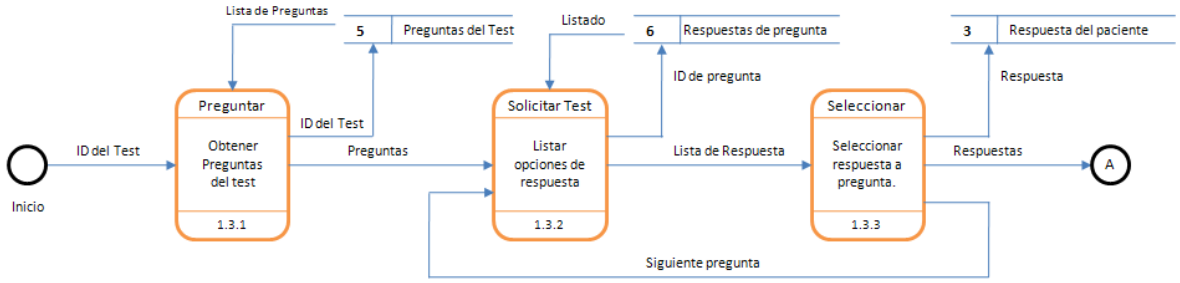


Diagrama de flujo de datos nivel 2.

1.4 Evaluar

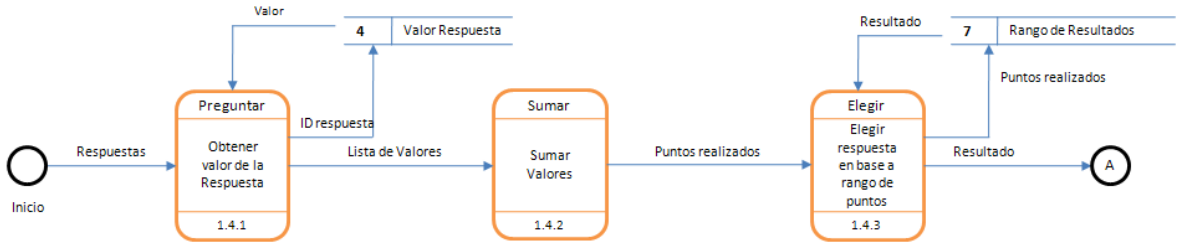
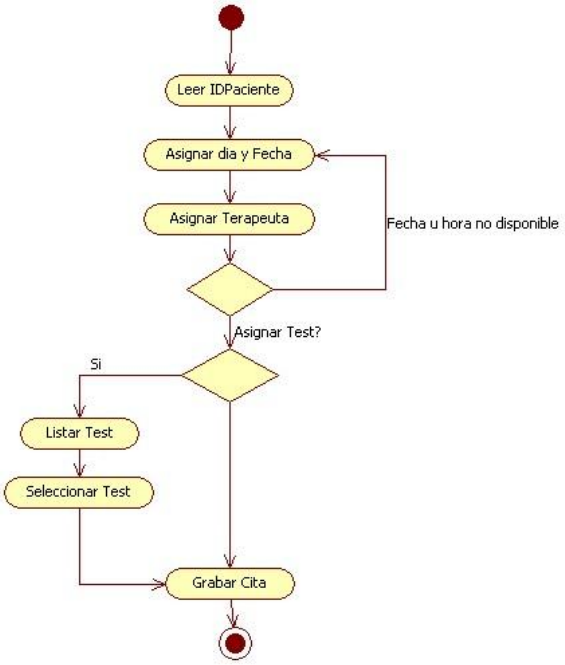


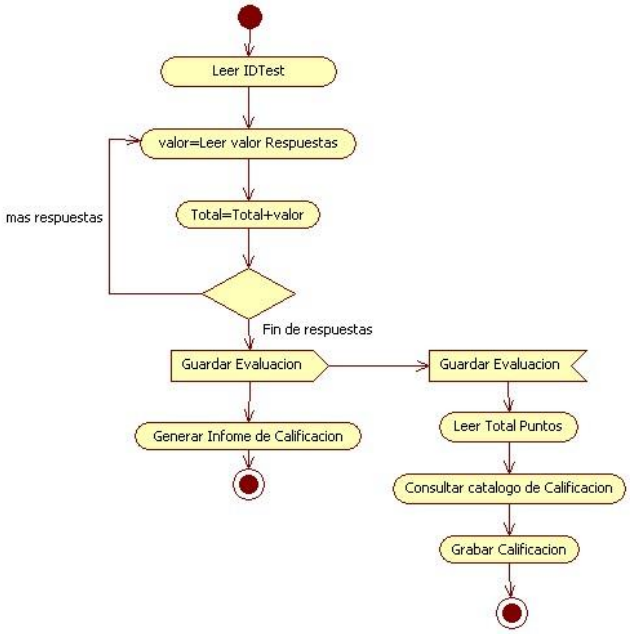
Diagrama de flujo de datos nivel 2.

11. Anexo C: Diagrama de Actividades.

Agenda Cita Paciente.



Evaluar Test.



Obtener Reactivos.

Contestar Reactivo.

12. Anexo D: Diccionario de datos.

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
1	PCNTE	ID_PCNTE	Identificador del Paciente	[A-Z]{6}AAMMDD	N/A	X	No	char	12	N/A	Se calcula mediante las primeras dos letras del apellido paterno, dos del apellido materno, dos del primer nombre y su fecha de nacimiento en formato AAMMDD, donde AA es año, MM mes y DD día. En caso de que no cuente con alguno de los dos apellidos, se colocará "XX"
2	PCNTE	SXO	Identificador del sexo del Paciente	[0,1]	0=Masculino; 1=Femenino;		No	bit	N/A	N/A	N/A
3	PCNTE	RFC	Registro Federal de Contribuyentes del Paciente	[A-Z]{4}AAMMDD([0-9][A-Z][0-9])?	N/A		No	char	13	N/A	N/A
4	PCNTE	CURP	Clave Única de Registro de Población del Paciente	[A-Z]{4}AAMMDD[A-Z]{6}[0-9]{2}	N/A		Si	char	18	N/A	N/A
5	PCNTE	FCHA_BDA	Fecha en la que se casó el Paciente, con ellos se calcula el número de años que lleva de casado, que es uno de los factores de análisis en el programa	AAAAMMDD	N/A		Si	date	N/A	N/A	Si no se cuenta con la fecha exacta, al menos se debe proporcionar el año y el mes y por default se pondrá el primer día del mes
6	PCNTE	OCPCION	Descripción de la ocupación del Paciente	Todos los valores	N/A		Si	varchar	100	N/A	N/A
7	PCNTE	LGAR_OCPCION	Descripción del sitio donde labora el Paciente	Todos los valores	N/A		Si	varchar	200	N/A	N/A
8	PCNTE	BNDRA_TRBJO	Indicador que permite saber si la actividad que	[0,1]	0=No es un trabajo remunerado;		No	bit	N/A	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			realiza el Paciente es remunerativa		1=Si es un trabajo remunerado;						
9	PCNTE	ANIOS_ESTDIO	Número de años máximos de estudio del Paciente de acuerdo a su grado de escolaridad	[0-9]{1-2}	N/A		No	numeric	2	0	N/A
10	PCNTE	MTVO	Descripción del motivo por el que el Paciente decidió entrar al programa	Todos los valores	N/A		No	varchar	1024	N/A	N/A
11	PCNTE	PSO	Número que indica el peso en kilogramos del Paciente	[0-9]{3}.[0-9]{2}	N/A		Si	numeric	5	2	N/A
12	PCNTE	RLGION	Descripción de la religión del Paciente	Todos los valores	N/A		Si	varchar	100	N/A	N/A
13	PCNTE	TLFNO1	Número telefónico principal del Paciente	Todos los valores	N/A		No	varchar	15	N/A	N/A
14	PCNTE	TLFNO2	Número telefónico secundario o auxiliar del Paciente que normalmente es el celular	Todos los valores	N/A		Si	varchar	15	N/A	N/A
15	PCNTE	FCHA_MDFCCION	Fecha de última modificación de datos del Paciente en el sistema	AAAAMMDD	N/A		No	date	N/A	N/A	Si cambia de valor en el sistema cualquiera de los siguientes campos, entonces se registra la fecha del sistema: OCPCION, LGAR_OCPCION, BNDRA_TRBJO, ANIOS_ESTDIO, PSO, RLGION, ID_ESTDO_CVIL, ID_ESCLRDAD, ID_SSTNCIA, ID_PRGRMA, ID_ETPA, ID_TPO_PCNTE

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
16	PCNTE	ID_ESTDO_CVIL	Código del estado civil del Paciente	[1-9]	1=Soltero(a);2=Casado(a);3=Unión Libre;4=Viudo(a);5=Divorciado(a);		No	numeric	1	0	N/A
17	PCNTE	ID_ESCLRDAD	Código del grado máximo de escolaridad del Paciente	[1-9]	1=Primaria;2=Secundaria;3=Bachillerato;4=Licenciatura;5=Maestría;6=Doctorado;		No	numeric	1	0	N/A
18	PCNTE	ID_SSTNCIA	Código de la sustancia que consume el Paciente	[1-9]	1=Alcohol;2=Marihuana;3=Cocaina;4=Benzodicepinas;5=Barbituricos;6=Poliusuario;		No	numeric	1	0	N/A
19	PCNTE	ID_MDIO	Código del medio por el que el Paciente se enteró del programa	[1-9]	1=Poster;2=Radio;3=TV;4=Servicios Médicos;5=Tríptico;6=Internet;		No	numeric	1	0	N/A
20	PCNTE	ID_PRGRMA	Código del programa de ayuda psicológica en el que participa el Paciente	([1-9],[1-9][0-9])	1=Programa de Satisfactores Cotidianos;		No	numeric	2	0	N/A
21	PCNTE	ID_ETPA	Código de la etapa en la que se encuentra el Paciente en el programa	([1-9],[1-9][0-9])	1=Primer Contacto;2=Admisión;3=Evaluación 1;4=Evaluación 2;5=Tratamiento;6=Seguimiento 1;		No	numeric	2	0	N/A
22	PCNTE	ID_TPO_PCNTE	Código del tipo de Paciente que define el origen de donde proviene	[1-9]	1=Personal Administrativo de la UNAM;2=Estudiante de la		No	numeric	1	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
					UNAM;3=Público en General;						
23	ESTDO_CVIL	ID_ESTDO_CVIL	Identificador para el catálogo de estado civil	[1-9]	1=Soltero(a);2=Casado(a);3=Unión Libre;4=Viudo(a);5=Divorciado(a);	X	No	numeric	1	0	N/A
24	ESTDO_CVIL	NMBRE	Nombre del estado civil para cada identificador en el catálogo	Todos los valores	N/A		No	varchar	15	N/A	N/A
25	ESCLRDAD	ID_ESCLRDAD	Identificador para el catálogo de escolaridad	[1-9]	1=Primaria;2=Secundaria;3=Bachillerato;4=Licenciatura;5=Maestría;6=Doctorado;	X	No	numeric	1	0	N/A
26	ESCLRDAD	NMBRE	Nombre de la escolaridad para cada identificador en el catálogo	Todos los valores	N/A		No	varchar	15	N/A	N/A
27	SSTNCIA	ID_SSTNCIA	Identificador para el catálogo de sustancias de consumo	[1-9]	1=Alcohol;2=Marihuana;3=Cocaina;4=Benzodicepinas;5=Barbituricos;6=Poliusuario;	X	No	numeric	1	0	N/A
28	SSTNCIA	NMBRE	Nombre de la sustancia para cada identificador en el catálogo	Todos los valores	N/A		No	varchar	15	N/A	N/A
29	MDIO	ID_MDIO	Identificador para el catálogo del medio en el que se enteran del centro de ayuda psicológica	[1-9]	1=Poster;2=Radio;3=TV;4=Servicios Médicos;5=Tríptico;6=Internet;	X	No	numeric	1	0	N/A
30	MDIO	NMBRE	Nombre del medio para cada identificador en el	Todos los valores	N/A		No	varchar	15	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			catálogo								
31	TPO_PCNTE	ID_TPO_PCNTE	Identificador para el catálogo de tipo de paciente de acuerdo a su origen	[1-9]	1=Personal Administrativo de la UNAM;2=Estudiante de la UNAM;3=Público en General;	X	No	numeric	1	0	N/A
32	TPO_PCNTE	NMBRE	Nombre del tipo de paciente para cada identificador en el catálogo	Todos los valores	N/A		No	varchar	20	N/A	N/A
33	ETPA	ID_PRGRMA	Código de cada programa impartido por el centro de ayuda psicológica contra adicciones Acasulco	([1-9],[1-9][0-9])	1=Programa de Satisfactores Cotidianos;	X	No	numeric	2	0	N/A
34	ETPA	ID_ETPA	Identificador de cada etapa que compone un programa impartido en el centro de ayuda psicológica Acasulco	([1-9],[1-9][0-9])	1=Primer Contacto;2= Admisión;3= Evaluación 1;4=Evaluación 2;5=Tratamiento;6=Seguimiento 1;	X	No	numeric	2	0	N/A
35	ETPA	NMBRE	Nombre de cada etapa por cada programa del centro Acasulco	Todos los valores	N/A		No	varchar	20	N/A	N/A
36	USRIO	ID_USRIO	Identificador de usuario con el que Pacientes, Terapeutas y el Administrador del sistema se identifica	Todos los valores	N/A	X	No	char	12	N/A	N/A
37	USRIO	CNTRSNIA	Contraseña con la que se identifica y valida cada usuario del sistema	Todos los valores	N/A		No	char	32	N/A	Cadena encriptada mediante el algoritmo de hash MD5

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
38	USRIO	NMBRE	Nombre propio del usuario del sistema	Todos los valores	N/A		No	varchar	80	N/A	N/A
39	USRIO	APLLDO_PTRNO	Apellido paterno del usuario del sistema	Todos los valores	N/A		Si	varchar	80	N/A	N/A
40	USRIO	APLLDO_MTRNO	Apellido materno del usuario del sistema	Todos los valores	N/A		Si	varchar	80	N/A	N/A
41	USRIO	CRREO_ELCTRNCO	Correo electrónico del usuario del sistema	Todos los valores de al menos 5 caracteres	N/A		Si	varchar	120	N/A	N/A
42	USRIO	FCHA_NCMNTO	Fecha de nacimiento del usuario del sistema	AAAAMMDD	N/A		No	date	N/A	N/A	N/A
43	USRIO	FCHA_ALTA	Fecha de alta del usuario del sistema	AAAAMMDD	N/A		No	date	N/A	N/A	Fecha del sistema al momento de registrar al usuario
44	USRIO	FCHA_MDFCCION	Fecha de última modificación de los datos del usuario del sistema	AAAAMMDD	N/A		No	date	N/A	N/A	Si cambia de valor en el sistema cualquiera de los siguientes campos, entonces se registra la fecha del sistema: ESTTUS ó ID_DRCCION
45	USRIO	ESTTUS	Indicador que permite definir si el usuario puede entrar o no al sistema, es decir si está activo o no	(0,1)	0=Inactivo;1=Activo;		No	bit	N/A	N/A	N/A
46	USRIO	ID_DRCCION	Código de la dirección del usuario del sistema	([1-9],[1-9][0-9]{1-10})	N/A		No	numeric	11	0	N/A
47	USRIO	ID_PRFIL	Código del perfil del usuario del sistema	([1-9],[1-9][0-9])	1=Administrador;2=Terapeuta;3=Paciente;		No	numeric	2	0	N/A
48	DRCCION	ID_DRCCION	Identificador de la dirección del usuario del sistema	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
49	DRCCION	CLLE	Nombre de la calle	Todos los	N/A		No	varchar	200	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			donde vive el usuario del sistema	valores							
50	DRCCION	NMRO_EXTRIOR	Número exterior donde vive el usuario del sistema	Todos los valores	N/A		No	varchar	30	N/A	N/A
51	DRCCION	NMRO_INTRIOR	Número interior donde vive el usuario del sistema	Todos los valores	N/A		Si	varchar	30	N/A	N/A
52	DRCCION	FCHA_MDFCCION	Fecha de modificación de la dirección donde vive el usuario el sistema	AAAAMMDD	N/A		No	date	N/A	N/A	N/A
53	DRCCION	ID_DRCCION_SEPOMEX	Código de la dirección de SEPOMEX (Servicio Postal Mexicano), donde se hace referencia a la localidad, estado, colonia y código postal	([1-9],[1-9][0-9]{1-10})	1=Localidad: Azcapotzalco, Estado: Distrito Federal, Colonia: Azcapotzalco Centro, C.P. 02000;2=Localidad: Azcapotzalco, Estado: Distrito Federal, Colonia: Delegación Política Azcapotzalco, C.P. 02008;3=Localidad: Azcapotzalco, Estado: Distrito Federal, Colonia: Los Reyes, C.P. 02010;		No	numeric	11	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
54	DRCCION_SEPOMEX	ID_DRCCION_SEPOMEX	Identificador de la dirección de SEPOMEX (Servicio Postal Mexicano)	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
55	DRCCION_SEPOMEX	ID_LCLDAD	Código de la localidad de acuerdo al identificador de la dirección de SEPOMEX (Servicio Postal Mexicano)	([1-9],[1-9][0-9]{1-5})	2=Azcapotzalco si el estado es 9: Distrito Federal;2=Acolman si el estado es 15: Estado de México;3=Coyoacan si el estado es 9: Distrito Federal;		No	numeric	5	0	N/A
56	DRCCION_SEPOMEX	ID_ESTDO	Código del estado de la republica de la dirección SEPOMEX (Servicio Postal Mexicano)	([1-9],[1-9][0-9])	9=Distrito Federal;15= Estado de México;		No	numeric	2	0	N/A
57	DRCCION_SEPOMEX	CLNIA	Nombre de la colonia de la dirección SEPOMEX (Servicio Postal Mexicano)	Todos los valores	N/A		No	varchar	100	N/A	N/A
58	DRCCION_SEPOMEX	CDGO_PSTAL	Número de código postal de la dirección SEPOMEX (Servicio Postal Mexicano)	[0-9]{5}	N/A		No	char	5	0	N/A
59	LCLDAD	ID_LCLDAD	Identificador de la Localidad (Delegación, Municipio o Población) de acuerdo a cada estado de la República	([1-9],[1-9][0-9]{1-5})	N/A	X	No	numeric	5	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			Mexicana								
60	LCLDAD	ID_ESTADO	Código identificador del estado de la República Mexicana al que pertenece la localidad	([1-9],[1-9][0-9])	9=Distrito Federal;15= Estado de México;	X	No	numeric	2	0	N/A
61	LCLDAD	NMBRE	Nombre de la localidad (Delegación, Municipio o Población) de acuerdo a cada estado de la República Mexicana	Todos los valores	N/A		No	varchar	60	N/A	N/A
62	ESTDO	ID_ESTDO	Identificador del estado de la República Mexicana	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
63	ESTDO	NMBRE	Nombre del estado de la República Mexicana	Todos los valores	N/A		No	varchar	30	N/A	N/A
64	FMLIAR	ID_PCNTE	Código Identificador del paciente que tiene parentesco con algún familiar	[A-Z]{6}AAMMDD	N/A	X	No	char	12	N/A	Se calcula mediante las primeras dos letras del apellido paterno, dos del apellido materno, dos del primer nombre y su fecha de nacimiento en formato AAMMDD, donde AA es año, MM mes y DD día. En caso de que no cuente con alguno de los dos apellidos, se colocará "XX"
65	FMLIAR	ID_FMLIAR	Identificador del familiar de cada paciente	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
66	FMLIAR	NMBRE	Nombre del familiar de cada paciente	Todos los valores	N/A		No	varchar	80	N/A	N/A
67	FMLIAR	APLLDO_PTRNO	Apellido paterno del familiar de	Todos los valores	N/A		Si	varchar	80	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			cada paciente								
68	FMLIAR	APLLDO_MTRNO	Apellido materno del familiar de cada paciente	Todos los valores	N/A		Si	varchar	80	N/A	N/A
69	FMLIAR	TLFNO1	Número de teléfono principal para localizar al familiar de cada paciente	Todos los valores	N/A		No	varchar	15	N/A	N/A
70	FMLIAR	TLFNO2	Número telefónico secundario o auxiliar del familiar de cada paciente que normalmente es el celular	Todos los valores	N/A		Si	varchar	15	N/A	N/A
71	FMLIAR	CRREO_ELCTRNCO	Correo electrónico del usuario del sistema	Todos los valores de al menos 5 caracteres	N/A		Si	varchar	120	N/A	N/A
72	FMLIAR	ID_TPO_FMLIAR	Código del tipo de familiar de cada paciente	[1-9]	1=Esposo(a); 2=Hijo(a);3= Abuelo(a);4= Tio(a);		No	numeric	2	0	N/A
73	FMLIAR	ID_DRCCION	Código de la dirección del familiar de cada paciente	([1-9],[1-9][0-9]{1-10})	N/A		No	numeric	11	0	N/A
74	TPO_FMLIAR	ID_TPO_FMLIAR	Identificador del tipo de familiar que puede tener una persona	[1-9]	N/A	X	No	numeric	2	0	N/A
75	TPO_FMLIAR	NMBRE	Nombre del tipo de familiar	Todos los valores	N/A		No	varchar	20	N/A	N/A
76	PRFIL	ID_PRFIL	Identificador del perfil del usuario del sistema	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
77	PRFIL	NMBRE	Nombre del perfil del usuario del sistema	Todos los valores	N/A		No	varchar	30	N/A	N/A
78	PRFIL	DSCRPCION	Descripción del perfil del usuario del sistema, donde se especifica con más detalle su rol en el sistema	Todos los valores	N/A		Si	varchar	100	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
79	EXPDNTE	ID_EXPDNTE	Identificador del expediente del paciente	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
80	EXPDNTE	ID_PCNTE	Identificador del paciente al que le pertenece el expediente	[A-Z]{6}AAMMDD	N/A		No	char	12	N/A	Se calcula mediante las primeras dos letras del apellido paterno, dos del apellido materno, dos del primer nombre y su fecha de nacimiento en formato AAMMDD, donde AA es año, MM mes y DD día. En caso de que no cuente con alguno de los dos apellidos, se colocará "XX"
81	EXPDNTE	ID_TRPTA	Identificador del terapeuta que atiene al paciente	[A-Z]{6}AAMMDD	N/A		No	char	12	N/A	Se calcula mediante las primeras dos letras del apellido paterno, dos del apellido materno, dos del primer nombre y su fecha de nacimiento en formato AAMMDD, donde AA es año, MM mes y DD día. En caso de que no cuente con alguno de los dos apellidos, se colocará "XX"
82	EXPDNTE	FCHA_MDFCCION	Fecha de última modificación del expediente del paciente	AAAAMMDD	N/A		No	date	N/A	N/A	Es la fecha del sistema al cambiar cualquiera de los siguientes campos de la tabla EXPEDNTE: ID_TRPTA, NMBRE_DRCTRIOu OBSRVCNES
83	EXPDNTE	NMBRE_DRCTRIO	Nombre del directorio del sistema que guarda documentos electrónicos adjuntos al expediente del paciente	Todos los valores	N/A		No	varchar	255	N/A	N/A
84	EXPDNTE	OBSRVCNES	Descripción de las últimas observaciones hechas por el	Todos los valores	N/A		Si	varchar	512	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			terapeuta al paciente sobre su estado general								
85	SSION	ID_EXPDNTE	Código Identificador del expediente del paciente asociado a una sesión en el centro de ayuda psicológica	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
86	SSION	ID_SSION	Identificador de la sesión dedicada al paciente para su tratamiento, esta asociada a su expediente	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
87	SSION	FCHA_PRGRMCION	Fecha en la que se programa la sesión al paciente	AAAAMMDD	N/A		No	date	N/A	N/A	Cuando un terapeuta programa una sesión al paciente en este campo se guarda la fecha del sistema
88	SSION	FCHA_INCIO	Fecha en la que debe iniciar la sesión con el paciente	AAAAMMDD	N/A		Si	date	N/A	N/A	N/A
89	SSION	FCHA_CNCLSION	Fecha en la que se concluye la sesión con el paciente	AAAAMMDD	N/A		Si	date	N/A	N/A	N/A
90	SSION	OBSRVCNES	Descripción de las observaciones hechas por el terapeuta al paciente para cada sesión	Todos los valores	N/A		Si	varchar	512	N/A	N/A
91	SSION	ID_TPO_SSION	Código del tipo de sesión programada al paciente	[1-9]	1=TERAPIA; 2=TEST;3=MIXTA;		No	numeric	1	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
92	SSION	ID_ESTTUS_SSION	Código del estatus de la sesión	[1-9]	1=EN TIEMPO;2=RETRASADA;3=CONCLUIDA;4=CANCELADA;		No	numeric	1	0	Si el campo FCHA_CNCLSION no se ha registrado (NULL) toma el valor de 1; si el campo FCHA_CNCLSION es mayor que el campo FCHA_INCIO entonces toma el valor de 2; si el campo FCHA_CNCLSION es igual que el campo FCHA_INCIO entonces toma el valor de 3; si el terapeuta decide cancelar la sesión toma el valor de 4
93	SSION	ID_USRIO	Código del usuario que programó la sesión al paciente	Todos los valores	N/A		No	char	12	N/A	N/A
94	ESTTUS_SSION	ID_ESTTUS_SSION	Identificador del estatus de la sesión	[1-9]	N/A	X	No	numeric	1	0	N/A
95	ESTTUS_SSION	NMBRE	Nombre del estatus de la sesión	Todos los valores	N/A		No	varchar	15	N/A	N/A
96	TPO_SSION	ID_TPO_SSION	Identificador del tipo de sesión	[1-9]	N/A	X	No	numeric	1	0	N/A
97	TPO_SSION	NMBRE	Nombre del tipo de sesión	Todos los valores	N/A		No	varchar	10	N/A	N/A
98	PRGRMA	ID_PRGRMA	Identificador del programa impartido por el centro de ayuda psicológica a sus pacientes	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
99	PRGRMA	NMBRE	Nombre del programa de ayuda psicológica	Todos los valores	N/A		No	varchar	100	N/A	N/A
100	PRGRMA	DSCRPCION	Descripción del programa de ayuda psicológica	Todos los valores	N/A		Si	varchar	1024	N/A	N/A
101	PRGRMA	ESTTUS	Indicador que define si el programa de ayuda psicológica esta vigente o no	(0,1)	0=No vigente;1=Vigente;		No	bit	N/A	N/A	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
102	PRGRMA_TEST	ID_PRGRMA	Código identificador del programa asociado a cada test psicológico que se puede aplicar durante la duración del mismo a un paciente	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
103	PRGRMA_TEST	ID_TEST	Identificador del test asociado a cada programa de ayuda psicológica	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
104	TEST_PRGRMDO	ID_PRGRMA	Código identificador del programa asociado al test programado	([1-9],[1-9][0-9])	1=Programa de Satisfactores Cotidianos;	X	No	numeric	2	0	N/A
105	TEST_PRGRMDO	ID_TEST	Código identificador del test que se ha programado	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
106	TEST_PRGRMDO	ID_EXPDNTE	Código identificador del expediente asociado al test programado	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
107	TEST_PRGRMDO	ID_SSION	Código identificador de la sesión donde se ha programado el test	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
108	TEST_PRGRMDO	ID_ETPA	Código identificador de la Etapa asociado al test programado	([1-9],[1-9][0-9])	1=Primer Contacto;2= Admisión;3= Evaluación 1;4=Evaluación 2;5=Tratamiento;6=Seguimiento 1;	X	No	numeric	2	0	N/A
109	TEST	ID_TEST	Identificador del test	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
110	TEST	NMBRE	Nombre del test	Todos los valores	N/A		No	varchar	100	N/A	N/A
111	TEST	INSTRCCNES	Instrucciones para contestar el test	Todos los valores	N/A		Si	varchar	1024	N/A	N/A
112	TEST	ESTTUS	Indicador que define si el test esta activo o no para ser programado	(0,1)	0=Inactivo;1=Activo;		No	bit	N/A	N/A	N/A
113	PRGNTA	ID_TEST	Código identificador del test al que pertenece la pregunta	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
114	PRGNTA	ID_PRGNTA	Identificador de la pregunta	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
115	PRGNTA	PRGNTA	Descripción de la pregunta	Todos los valores	N/A		No	varchar	100	N/A	N/A
116	PRGNTA	ESTTUS	Indicador que define si la pregunta esta activa o no para el test	(0,1)	0=Inactiva;1=Activa;		No	bit	N/A	N/A	N/A
117	PRGNTA	ID_TPO_PRGNTA	Código del tipo de pregunta que determina cómo se mostrará el grupo de respuestas en el cuestionario	([1-9],[1-9][0-9])	1=Libre;2=Checkbox;3=Select;		No	numeric	2	0	N/A
118	TPO_PRGNTA	ID_TPO_PRGNTA	Identificador del tipo de pregunta que determina cómo se mostrará el grupo de respuestas en el cuestionario	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
119	TPO_PRGNTA	NMBRE	Nombre del tipo de pregunta	Todos los valores	N/A		No	varchar	30	N/A	N/A
120	PRGNTA	ID_CLSFCCION	Código del tipo de clasificación o característica que se evalúa al responder la misma	([1-9],[1-9][0-9])	1=Dependencia a la sustancia;2=Calidad de vida;3=Entorno Familiar;		No	numeric	2	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
121	CLSFCCION	ID_CLSFCCION	Identificador del tipo de clasificación o característica a evaluar	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
122	CLSFCCION	NMBRE	Nombre del tipo de clasificación o característica a evaluar	Todos los valores	N/A		No	varchar	30	N/A	N/A
123	OPCION	ID_TEST	Código identificador del test asociado a la opción	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
124	OPCION	ID_PRGNTA	Código identificador de la pregunta asociada a cada opción	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
125	OPCION	ID_OPCION	Identificador de la opción	([0-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
126	OPCION	OPCION	Descripción de la opción que se le muestra al paciente que va a responder el test	Todos los valores	N/A		No	varchar	80	N/A	N/A
127	OPCION	VLOR	Valor que puede adquirir la opción en caso de ser respondida por el paciente según las condiciones de diseño del test	([1-9],[1-9][0-9]{1-8}).([1-9],[1-9][0-9])	N/A		No	numeric	11	2	N/A
128	OPCION	LNGTUD	Número de caracteres para desplegar las opciones en el sistema	([1-9],[1-9][0-9]{1-10})	N/A		Si	numeric	11	0	N/A
129	OPCION	SLTAR	Número de pregunta a saltar en caso de que se haya elegido la opción asociada como respuesta	([1-9],[1-9][0-9]{1-10})	N/A		Si	numeric	11	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
130	OPCION	AUXLIAR	Identificador para mostrar el cuadro de texto de la respuesta libre en el sistema	(0,1)	0=No mostrar;1=Mostrar;		Si	char	1	N/A	N/A
131	OPCION	ESTTUS	Indicador que define si la opción es elegible para ser mostrada al paciente durante el test	(0,1)	0=No elegible;1=Elegible;		No	bit	N/A	N/A	N/A
132	RSPSTA	ID_EXPDNTE	Código identificador del expediente del paciente asociado a cada respuesta	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
133	RSPSTA	ID_ETPA	Código identificador de la Etapa asociado al test programado	([1-9],[1-9][0-9])	1=Primer Contacto;2=Admisión;3=Evaluación 1;4=Evaluación 2;5=Tratamiento;6=Seguimiento 1;	X	No	numeric	2	0	N/A
134	RSPSTA	ID_SSION	Código identificador de la sesión asociada a cada respuesta del paciente	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
135	RSPSTA	ID_PRGRMA	Código identificador del programa al que pertenece el test asociado a cada respuesta del paciente	([1-9],[1-9][0-9])	N/A	X	No	numeric	2	0	N/A
136	RSPSTA	ID_TEST	Código identificador del test que responde el paciente asociado a cada respuesta	([1-9],[1-9][0-9]{1-10})	N/A	X	No	numeric	11	0	N/A
137	RSPSTA	ID_PRGNTA	Código identificador de la	([1-9],[1-9][0-9]{1-	N/A	X	No	numeric	11	0	N/A

Id	Entidad	Nombre	Descripción	Dominio	Códigos	Llave	Nulo	Tipo de dato	Longitud	Decimales	Cálculo
			pregunta respondida por el paciente en el test	10})							
138	RSPSTA	ID_OPCION	Código identificador de la opción elegida como respuesta por el paciente	([0-9],[1-9][0-9]{1-10})	N/A		No	numeric	11	0	N/A
139	RSPSTA	RSPSTA_LIBRE	Descripción de la respuesta libre del paciente en caso de aplicar	Todos los valores	N/A		Si	varchar	512	N/A	N/A

13. GLOSARIO.

- ***AJAX***: JavaScript asíncrono y XML. Técnica de desarrollo WEB para crear aplicaciones interactivas o RIA ((Aplicaciones de Internet Enriquecidas).
- ***Aplicación WEB***: Aplicación que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador.
- ***API***: Interfaz de Programación de Aplicaciones provista por los creadores del lenguaje Java y que da a los programadores los medios para desarrollar aplicaciones Java.
- ***Base de Datos***: Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
- ***Caso de Uso***: Técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software.
- ***Clase***: Definición de un conjunto de objetos que comparten una estructura y comportamiento.
- ***Compilador***: Programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.
- ***Hardware***: Conjunto de todas las partes físicas y tangibles de una computadora: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos; sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado; contrariamente al soporte lógico e intangible que es llamado software.
- ***HTTP***: Protocolo para la Transferencia de HiperTexto.
- ***Ingeniería de Software***: Rama de la ingeniería que aplica los principios de las ciencias de la computación y las matemáticas para lograr resultados eficaces en los desarrollos de software.
- ***Internet:*** Conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

- ***Intranet***: Red de computadoras privadas que utiliza tecnología Internet para compartir de forma segura cualquier información o programa del sistema operativo para evitar que cualquier usuario de Internet pueda ingresar

- ***Java***: Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

- ***JDBC***: Conectividad de Base de Datos Java más conocio por sus siglas JDBC, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

- ***JVM***: Máquina Virtual Java. Programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

- ***Lenguaje de Programación***: Conjunto de signos y reglas que permite la comunicación con un ordenador o computadora.

- ***MVC***: Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario,y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

- ***Objeto***: Entidad que combinan estado, comportamiento e identidad. El estado está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos)., el comportamiento está definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él y la identidad es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

- ***Paradigma de Programación***: Colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan, al final la estructura de un programa.

- ***Patrón de Diseño***: Solución más conveniente a un problema común de programación.

- ***Procedimiento Almacenado***: Programa (o procedimiento) el cual es almacenado físicamente en una base de datos. Su implementación varía de un manejador de bases de datos a otro. La ventaja de un procedimiento almacenado es que al ser ejecutado en

respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado.

- ***Programación Orientada a Objetos***: Técnica o estilo de programación que utiliza objetos como bloque esencial de construcción.
- ***Servidor Web***: Programa que implementa el protocolo HTTP para transferir documentos.
- ***Sistema Informático***: Conjunto de partes interrelacionadas, hardware, software y de Recurso Humano (humanware). Un sistema informático típico emplea una computadora que usa dispositivos programables para capturar, almacenar y procesar datos.
- ***Sistema Operativo***: Conjunto de programas de computación destinados a realizar muchas tareas entre las que destaca la administración eficaz de sus recursos.
- ***Software***: Conjunto de los programas de cómputo, procedimiento, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.
- ***SQL***: Lenguaje de consulta estructurado. Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas.
- ***UML***: Lenguaje Unificado de Modelado.
- ***ZK***: Framework de aplicaciones web en AJAX, completamente en Java de Código abierto que permite una rica interfaz de usuario para aplicaciones web sin usar JavaScript y con poca programación.
- ***JavaScript***: Lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

9. REFERENCIAS

- ✓ *Piensa en Java*, Prentice Hall. Bruce Eckel, Segunda Edición.
- ✓ *The Java™ Programming Language, Fourth Edition,* Ken Arnold, James Gosling, David Holmes. Addison Wesley Professional
- ✓ *SCJP exam for J2SE 5: a concise and comprehensive study guide for the Sun certified java programmer exam*. Paul Sanghera. Berkeley, California.
- ✓ *Aprendiendo UML en 24 Horas,* Prentice Hall, Joseph Schmuler, 1a Edición
- ✓ *Ingeniería de software orientado a objetos*, Bernd Bruegge, Allen H. Dutoit. Pearson Educación, 2002
- ✓ *Introducción a las bases de datos. El Modelo Relacional*. Olga Pons Capote; Nicolás Maarín Ruiz; Juan Miguel Medina Rodríguez ; Et. At. Editorial Thomson.

Recursos Web:

- ✓ 12 Reglas de Cod:

  http://es.wikipedia.org/wiki/12_reglas_de_Codd
- ✓ UML :

  http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/what_is_uml.htm
- ✓ Framework ZK.

  http://www.zkoss.org
- ✓ Java

  http://java.sun.com
- ✓ Manual de referencia MYSQL

  http://downloads.mysql.com/docs/refman-5.0-es.chm