Universidad de las Ciencias Informáticas

Facultad 7

Título: Sistema para la gestión de la atención a pacientes en los cuerpos de guardia

Trabajo de Diploma para optar por el título de

Ingeniero en ciencias Informáticas

Autor(es): Ferniel Hernández González

Tutor(es): Yubismel Perdomo Velásquez

Asesor(a): Yudary Rojas Molina

Junio 2007

“Invertir en conocimientos produce siempre los mejores beneficios.”

Benjamin Franklin.

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Ferniel Hernández González **Yubismel Perdomo Velásquez**

_____ _____

Firma del Autor Firma del Tutor

DATOS DE CONTACTO

Tutor:

Yubismel Perdomo Velásquez: Profesor graduado de Ing. Informático en el año 2006. Ha impartido asignaturas como Introducción a la programación, Programación II y Sistemas Gestores de Bases de Datos. Posee categoría docente de adiestrado y actualmente cursa el diplomado de Docencia Universitaria. Correo electrónico: yubismel@uci.cu.

Asesora:

Yudary Rojas Molina: Licenciada en Letras (Filología) por la Universidad Central de las Villas, profesora instructora recién graduada. Correo electrónico: yudany@uci.cu.

AGRADECIMIENTOS

A mis padres que siempre me enseñan la importancia del esfuerzo para lograr las metas.
A nuestra Revolución por ofrecernos la oportunidad de poder realizar mis sueños en esta maravillosa universidad.
A todas las personas que de una u otra forma han contribuido a la realización de este trabajo.
A mis familiares, por ser esa fuente tan grande de apoyo incondicional.
A todos los profesores que dieron lo mejor de sí para convertirme en lo que soy.
A todos mis compañeros de estudio que también han ayudado al desarrollo de este trabajo.
A todos, les agradezco de todo corazón por haber hecho posible este sueño.

Muchas Gracias.

DEDICATORIA

A mi mamá, por ser la persona que me dió la vida.

A Nory, por sus consejos, amor y dedicación.

A Ferro, por brindarme tanto apoyo.

A mi abuela, que con su gran cariño me ha ayudado a llegar tan lejos.

A Norma, por tanta ayuda.

A mis amigos Reinier y Frank.

A todos los que forman parte de este logro.

RESUMEN

Actualmente, en los cuerpos de guardia de los hospitales cubanos, no existe un sistema informático que gestione la información acerca de la atención al paciente, por lo que todos estos procesos se realizan de forma manual trayendo consigo demora y dificultades para el control de esta información. Para solucionar este problema, se plantea como objetivo del presente trabajo, implementar una aplicación Web, para poder manejar toda la información correspondiente al cuerpo de guardia de un hospital.

Las principales herramientas y tecnologías utilizadas para la implementación del sistema son la plataforma .Net, C# como lenguaje desarrollador del negocio, AJAX y arquitectura en tres capas. Se utilizó como paradigma la programación orientada a objeto.

Se espera que el sistema propuesto sea utilizado en los centros hospitalarios del país, optimizando los servicios ofrecidos a los pacientes y contribuyendo a una mejor organización de la información en los cuerpos de guardia.

PALABRAS CLAVE

Rol de implementador, gestión hospitalaria, cuerpo de guardia.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El siglo XX, ha sido llamado por muchos científicos el “siglo de la informatización y la tecnología” puesto que el impacto de estas, ha repercutido en todas las esferas de la vida, de ahí la gran variedad de sistemas informáticos, que contribuyen a mejorar el trabajo en todos estos sectores de la sociedad.

En Cuba, existe un modelo para la informatización de la sociedad, compuesto por un conjunto de programas que desarrollan las estructuras técnicas a partir de las telecomunicaciones y de aplicaciones que facilitan el trabajo en todos los servicios, estas innovaciones tecnológicas se producen para beneficio de toda la sociedad.

El desarrollo de la informática, ha permitido la creación de herramientas y tecnologías que puestas al servicio de la salud, permiten que esta esfera sea cada día más productiva y capaz de asegurar mayor calidad en los servicios.

Dentro del área de la salud se llevan a cabo muchos procesos que actualmente necesitan de un soporte informatizado para optimizar el rendimiento de sus operaciones y por ende aumentar la calidad de atención a los pacientes.

Estas nuevas tecnologías vinculadas al sector de la salud ofrecen numerosas propuestas que garantizan el éxito en la informatización del proceso de dirección y organización de la información que se maneja y procesa dentro de un hospital.

El sistema de salud cubano se ha desarrollado notablemente, todo puesto que la Revolución le otorga a esta rama un carácter primordial. En la actualidad, existen problemas en la gestión del trabajo, debido a la vinculación entre varios servicios que se

prestan, a esto se le incluye que todas las tareas se realizan y archivan en papel. A raíz de toda esta problemática, se ha estudiado muchas veces la posibilidad de informatizar todos los servicios hospitalarios, y aunque se han hecho intentos, nunca se ha llegado a un producto que solucione la situación existente en todos los servicios.

El presente trabajo abordará las cuestiones principales del trabajo dentro de los hospitales. Dentro de este amplio marco de la atención hospitalaria, se centrará en el flujo de trabajo del módulo de cuerpo de guardia.

La tarea consiste en desarrollar una aplicación que gestione el proceso de atención a los pacientes en los cuerpos de guardia dentro del hospital. Además, de llevar un control de los recursos necesarios para poder llevar a cabo este proceso.

La mayoría de los servicios que brinda el sistema de salud no están automatizados. La situación problémica comienza en la gestión a la atención de los pacientes en el cuerpo de guardia, dado por el personal de guardia ese día, toda esta información relacionada se maneja en papel y todas estas actividades se realizan de forma manual o mediante algún software no estándar. Estos problemas pueden provocar que muchas veces se atrase el desempeño interno de los trabajadores, creándose una demora en la atención médica a la población.

Cuando un paciente llega al cuerpo de guardia, debe ser atendido según el nivel de gravedad con que se clasifique por parte del recepcionista, en la actualidad la determinación de que paciente pasa a la consulta se hace de manera manual. Esto trae

descoordinación y en muchas ocasiones pueden provocar una espera innecesaria de pacientes, cuyo tiempo de atención debe ser minimizado.

Los datos de los pacientes que son atendidos en las consultas de los cuerpos de guardia se registran en formularios existentes en formato de papel, a esto se le llama hoja de cargo médica que posee cada médico que brinda consulta. Posteriormente son enviados al terminar un turno de guardia a estadística, donde se archivan de forma física. Esta forma de guardar información no es persistente, resulta insegura, ya que pueden ocurrir pérdidas e inconsistencias en los datos. Además, los locales de archivos se tornan gigantes, situación no favorable para lograr un rápido y eficaz sondeo de alguna información. La redundancia de información trae consigo que en los últimos datos registrados a un paciente no existan los datos de las anteriores consultas hechas ese mismo día en el cuerpo de guardia, debido a que se crea uno nuevo, esto impide el seguimiento del paciente en relación a alguna patología, incluyendo la pérdida de tiempo al llenar nuevamente los mismos formularios.

Al no tener el médico la historia clínica a mano, durante la consulta no puede ver el historial del paciente y tiene que depender de la información que le brinden. En muchos los casos, los pacientes no la ofrecen, ya sea por desconocimiento o por falta de interés, y verdaderamente resulta importante para la realización de la consulta y la emisión de un buen diagnóstico.

Al tener cada servicio del departamento de guardia, un tipo de formulario para el registro de los datos, y un paciente puede pasar por varios servicios, ocurre que las

referencias registradas de un mismo paciente son tomados varias veces durante un mismo turno de guardia; por lo que estos procedimientos crean duplicados de la información por el desconocimiento de la existencia de la misma debido al gran torrente de datos que se opera, sin ninguna tecnología que auxilie dichos procesos.

Las grandes desventajas que traen consigo la realización de los procesos manuales para los trabajadores de la salud son una realidad que debe transformarse con la introducción de las tecnologías. La persistencia de registros actualizados se ve afectada a causa de la demora del llenado de datos y a los errores que se pueden cometer durante este procedimiento; de igual manera la cantidad de personas atendidas diariamente es reducida en comparación a lo que se puede llegar con una tecnología que cubra estos procesos.

Por lo anteriormente expuesto, podemos plantear que el problema actual que presenta el área de cuerpo de guardia es que no existe una aplicación que satisfaga todas sus necesidades.

Problema Científico:

¿Cómo erradicar los problemas en la gestión de la atención en los cuerpos de guardia de los hospitales cubanos?

Objeto de Estudio:

Procesos de la automatización de los servicios en los hospitales.

Objetivo de la Investigación:

Implementar una aplicación web para poder manejar toda la información correspondiente al cuerpo de guardia de un hospital.

Campo de acción:

Procesos de gestión de la atención médica automatizada de los cuerpos de guardia en los hospitales cubanos.

Objetivo General:

- Implementar una aplicación web para poder manejar toda la información correspondiente al cuerpo de guardia de un hospital.

Objetivo Específicos:

- Implementar una herramienta que pueda manejar todos los datos de los pacientes después de ser atendidos en el Cuerpo de Guardia.
- Investigar acerca de las características de las aplicaciones web.
- Investigar como se realizan los procesos vinculados con la gestión y atención a los pacientes en los cuerpos de guardia en los hospitales cubanos.
- Investigar acerca del uso y aplicación de los patrones de diseños en un software.
- Estudiar otros sistemas informáticos para la gestión en cuerpo de guardia.

Tareas de Investigación:

- Indagar acerca de las características de las aplicaciones web.
- Investigar como se realizan todos los procesos vinculados con la gestión y atención a los pacientes en los cuerpos de guardia en los hospitales cubanos.
- Investigar acerca del uso y aplicación de los patrones de diseños en un software.
- Estudiar otros sistemas informáticos para la gestión en cuerpo de guardia.

El presente trabajo consta de tres capítulos, en el primero, fundamentación teórica se hace una descripción de las tendencias y tecnologías actuales que se utilizaron como soporte para el desarrollo de la propuesta.

En el segundo análisis de la propuesta, se hace una valoración crítica del diseño propuesto por el analista del sistema, también se abordan temas como los patrones usados para el desarrollo de esta aplicación, los otros módulos con los que interactúa el cuerpo de guardia y sus estrategias de integración, y por último se hace una descripción de las principales clases necesarias para la implementación.

En el tercer capitulo validación de la propuesta, se realizan una serie de pruebas al software ya sea al prototipo funcional como al código.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En estos momentos, en el país se encuentran en pruebas, varios módulos del Sistema de Informatización Hospitalaria, pero son incompatibles con otros centros, pues poseen características propias de los lugares en donde se encuentran.

En este capítulo, se aborda los principales conceptos asociados, a los procesos de la automatización de los servicios en los hospitales, así como un estudio de las características fundamentales de algunos sistemas que existen en el mundo, similares a la propuesta, con el objetivo de permitir una mejor orientación hacia el objeto de desarrollo y fundamentar la necesidad de un software para la gestión de la atención medica dentro de los cuerpos de guardia en los hospitales; estos se basan fundamentalmente en la atención a los pacientes que acuden a ellos.

1.1 Tecnologías y sus tendencias

En la actualidad, la tendencia en el desarrollo de software está lejos de las grandes computadoras y enfocada hacia las estaciones de trabajo como plataformas de ingeniería del software. Las estaciones de trabajo individuales se interconectan mediante redes para que los ingenieros de software puedan comunicarse de forma efectiva. La base de datos de proyectos está disponible a través de un servidor de archivos en red que es accesible desde todas las estaciones de trabajo. Un sistema operativo que gestiona el hardware, la red y las herramientas mantiene todo el entorno unido.

En este capítulo se realizará un análisis profundo de las herramientas y las tecnologías a usar para la realización de este trabajo de diploma, estas serán las más adecuadas para darle una solución óptima al problema que se quiere resolver.

1.1.1 Aplicaciones web

El sistema que se desarrolla en este trabajo se basa en este tipo de aplicaciones ya que estas ofrecen muchas más ventajas que las aplicaciones de escritorio. La instalación de una aplicación Web se basa principalmente en la configuración de los componentes del lado del servidor, por lo que no es necesaria una instalación del lado del cliente, el accedería a ella mediante un navegador.

Principales ventajas:

- Para su uso basta con su instalación en un servidor, ya que el cliente puede acceder a ella mediante un navegador.
- Alta disponibilidad, ya que puede realizar consultas en cualquier parte del mundo donde tenga acceso a Internet y a cualquier hora.
- Se reduce el costo de mantenimiento debido a que el mantenimiento solo debe darse en el servidor.
- Son compatibles con cualquier sistema operativo.

1.1.2 AJAX

AJAX (Asynchronous JavaScript And XML) es un conjunto de tecnologías de desarrollo web para crear aplicaciones interactivas sin la necesidad de cargar la página completamente entre una solicitud y otra. Estas se ejecutan en el navegador, y mantiene comunicación asíncrona con el servidor en segundo plano mediante un canal de conexión. Debido a que la página web no se refresca completamente, si no solamente se acceden a elementos de la misma mediante JavaScript u otro lenguaje del lado del cliente, se gana en rapidez e interactividad.

Dado que las aplicaciones web tradicionales, envían formulario llenados por un usuario a un servidor web. Este responde enviando hacia atrás otra página web, por esta razón que en el servidor debe enviar una página a la vez las aplicaciones corren más

lentas. Sin embargo AJAX pueden enviar peticiones al servidor y solo traer las respuestas necesarias. Utilizando para ellos scripts, se procesa la respuesta del servidor.

El resultado de usar AJAX es muy factible pues resulta una aplicación con una interfaz con mayor respuesta, dado la cantidad de datos intercambiados entre el cliente y el servidor ya que es reducida grandemente por lo antes dicho. También se ahorra mucho tiempo de procesamiento en el servidor web ya que una gran parte de dicho procesamiento se realiza en el cliente.

1.1.3 Framework .NET

La plataforma Microsoft .NET proporciona todas las herramientas y tecnologías necesarias para desarrollar aplicaciones web distribuidas. Expone un modelo de programación para aplicaciones multinivel consistente e independiente del lenguaje de programación, a la vez que proporciona una interoperabilidad y una fácil migración desde las tecnologías actualmente existentes.

El .NET Framework es un entorno de desarrollo y ejecución multilenguaje de componentes que proporciona los bloques básicos para desarrollar aplicaciones web y servicios web. A grandes rasgos, está formado por un runtime universal (Common Language Runtime, o CLR), una librería de clases unificada común (Base Class Library) y ASP.NET, con formularios web, servicios web y servicios de acceso a datos. El Common Language Runtime (CLR) es un motor de ejecución de código de alto rendimiento. El código destinado a ejecutarse sobre el runtime se denomina código gestionado. Cuando un componente se ejecuta, el runtime es el responsable de crear objetos, invocar llamadas a métodos, gestionar la reserva de memoria (incluyendo la recolección de basura), iniciar y detener hilos y procesos, garantizar la política de seguridad y satisfacer cualquier dependencia que el componente tenga sobre otros

componentes, con independencia del lenguaje de programación en el que estén escritos.[1]

El .NET Framework también incluye un conjunto de librerías de clases (APIs) extensible, unificado, orientado a objetos y jerárquico que los desarrolladores pueden utilizar; la principal ventaja es que ya no necesitarán aprender múltiples librerías incompatibles de lenguajes independientes para realizar el trabajo. Al crear un conjunto de clases común entre todos los lenguajes de programación, es posible implementar herencia, gestión de errores y depuración cruzada multilenguaje.

El .NET Framework proporciona tanto clases abstractas como implementaciones de clases derivadas de esas clases básicas. Como desarrolladores, podemos utilizar estas clases derivadas directamente o derivar nuestras propias clases de ellas.

ASP.NET es precisamente el entorno de programación sobre el CLR y las librerías de clases básicas para desarrollar potentes aplicaciones web. Los modelos de programación basados en formularios web proporcionan un modo fácil y potente de crear interfaces de usuario dinámicos, y los servicios web de ASP.NET ofrecen los bloques básicos para construir aplicaciones web distribuidas que utilizan estándares como XML.

ASP. NET es un conjunto de tecnologías de desarrollo de aplicaciones web dinámicas comercializado por Microsoft. Es usado por programadores para construir sitios web domésticos, aplicaciones web y servicios XML. Forma parte de la plataforma .NET de Microsoft y es la tecnología sucesora de la tecnología Active Server Pages (ASP).

Soportado por un gran número de lenguajes, herramientas de desarrollo, la programación web es una mezcla de varios lenguajes de etiquetas, un gran uso de scripting y plataformas de servidor. Desafortunadamente para el programador de nivel intermedio, el conocimiento y habilidades que se necesitan para desarrollar aplicaciones web tienen muy poco en común con las que son necesarias en el desarrollo tradicional

[1] Ver en el anexo No. 8

de aplicaciones. Permite a los desarrolladores escribir código más limpio y más fácil de reutilizar y compartir, incrementando el rendimiento y la escalabilidad al poder acceder a lenguajes compilados, no interpretados.

Ventajas de la tecnología ASP.NET:

- Rendimiento: la aplicación se compila una sola vez al lenguaje nativo y luego, en cada petición tiene una compilación “Just in time”, quiere decir que se compila desde el código nativo, lo que permite mucho mejor rendimiento. También permite el almacenamiento del caché en el servidor.
- Rapidez en programación: mediante diversos controles, se puede con unas pocas líneas y en menos de 5 minutos mostrar toda una base de datos y hacer rutinas complejas.
- Formularios web: proporcionan un rico conjunto de servicios para ayudar a los desarrolladores a escribir páginas web interactivas.
- Servicios web: trae herramientas para compartir datos e información entre distintos sitios.
- Implantación: permite implantar una aplicación web simplemente copiando directorios.
- Seguridad: tiene diversas herramientas que garantizan la seguridad de nuestras aplicaciones.

ASP.NET es una tecnología realmente interesante. Es un nuevo paradigma completo que cambiará la forma de pensar sobre el desarrollo de software. Es una nueva forma de ver que el API de Windows, MFC, ATL y cualquier otra herramienta en la que se haya tenido que confiar dejan de tener el papel relevante que tenían y dan paso a una nueva librería de clases, incorporada como parte del .NET Framework. ¿Qué se gana? Un modelo de programación más unificado, la mejora de la seguridad y una nueva forma de escribir potentes aplicaciones web completas. Y esto solamente para los principiantes. ASP ha sido rediseñado partiendo de cero, el resultado es ASP.NET. Con un conjunto de características nuevas, ofrece código más fácil de escribir, reutilizar y compartir. ASP.NET mejora el rendimiento y escalabilidad ofreciendo acceso a lenguajes

compilados; el desarrollo es más intuitivo gracias a los formularios web, y su base orientada a objetos facilita la reutilización. Se soportan eventos, controles y funciones de caché. Los controles web, las técnicas de enlace de datos, los formularios web y los servicios web permiten sacar partido a las librerías de clases del .NET framework y permiten exponer funciones de negocio a través de la web, ofreciendo nuevas oportunidades de desarrollo.

1.2 Arquitectura utilizada

La arquitectura definida por los arquitectos de proyectos a utilizar es en tres capas. Esta arquitectura fue definida como una forma de organizar las capas jerárquicamente tal que cada capa pueda brindar servicios a la capa inmediatamente superior y a su vez esta se favorece de las prestaciones que le brinda la inmediatamente inferior. Mediante el llamado a métodos es como se ven estas interacciones entre las capas. Esta arquitectura facilita la modularidad del sistema así como corrección de errores y mejora el soporte del sistema [LANE, RECHTIN, BHANSALI, GARLAN, PERRY, CREMEN, 2003: http://www.sei.cmu.edu/].

En una arquitectura Cliente/Servidor se suelen utilizar dos capas, en este caso se está trabajando en el desarrollo de aplicaciones informáticas de gestión, esto se suele traducir en un servidor de bases de datos en los que se almacenan los datos y una aplicación cliente que contiene la interfaz de usuario y la lógica de la aplicación.

Mantener la misma arquitectura y pasar la lógica de la aplicación al servidor tampoco resulta una solución demasiado acertada. Se puede implementar la lógica de la aplicación utilizando procedimientos almacenados, pero estos suelen tener que implementarse en lenguajes estructurados no demasiado versátiles. Además, suelen ser lenguajes específicos para cada tipo de base de datos, por lo que la portabilidad del sistema se ve gravemente afectada.

Por lo tanto, se definió como una arquitectura que utilizaría tres capas lo cual facilitaría el trabajo así como quedaría como producto final un sistema con mayor portabilidad. Las tres capas son las siguientes:

- La capa de presentación o interfaz: es la encargada de interactuar directamente con el usuario, en este caso mediante una interfaz web.
- La capa de la lógica de la aplicación: es la que posee toda la lógica del negocio, usualmente implementada utilizando un modelo orientado a objetos del dominio de la aplicación, es la responsable de realizar todas las tareas para las cuales está diseñado el sistema.
- La capa de acceso a datos: es la encargada de gestionar todo el almacenamiento de los datos. Se refiere todos los procedimientos de almacenado que están en el gestor de base de datos.

1.3 Paradigma de programación orientada a objetos

La programación orientada a objetos como paradigma “es una forma de pensar, una filosofía, de la cual surge una cultura nueva que incorpora técnicas y metodologías diferentes. Pero estas técnicas y metodologías, y la cultura misma, provienen del paradigma, no lo hacen. La POO como paradigma es una postura ontológica: el universo computacional está poblado por objetos, cada uno responsabilizándose por sí mismo, y comunicándose con los demás por medio de mensajes"[GREIFF W. R., 1994: p. 31-39].

Este paradigma ya no se aplica solamente a los lenguajes de programación, además se viene aplicando en el análisis y diseño con mucho éxito, al igual que en las bases de datos. Para realizar una buena programación orientada a objetos hay que desarrollar todo el sistema aplicando esta tecnología, de ahí la importancia del análisis y el diseño orientado a objetos.

La programación orientada a objetos es una de las formas más populares de programar y viene teniendo gran acogida en el desarrollo de proyectos de software

desde los últimos años. Esta acogida se debe a sus grandes capacidades y ventajas frente a las antiguas formas de programar.

Tradicionalmente, la programación fue hecha en una manera secuencial o lineal, es decir una serie de pasos consecutivos con estructuras consecutivas y bifurcaciones. Los lenguajes basados en esta forma de programación ofrecían ventajas al principio, pero el problema ocurre cuando los sistemas se vuelven complejos. Estos programas escritos al estilo “espagueti” no ofrecen flexibilidad y el mantener una gran cantidad de líneas de código en sólo bloque se vuelve una tarea complicada.[2]

Frente a esta dificultad aparecieron los lenguajes basados en la programación estructurada. La idea principal de esta forma de programación es separar las partes complejas del programa en módulos o segmentos que sean ejecutados conforme se requieran. De esta manera se tiene un diseño modular, compuesto por módulos independientes que puedan comunicarse entre sí. Poco a poco este estilo de programación fue reemplazando al estilo “espagueti” impuesto por la programación lineal.

Entonces, se puede ver que la evolución que se fue dando en la programación se orientaba siempre a ir descomponiendo más el programa. Este tipo de descomposición conduce directamente a la programación orientada a objetos.

Pues la creciente tendencia de crear programas cada vez más grandes y complejos llevó a los desarrolladores a crear una nueva forma de programar que les permita crear sistemas de niveles empresariales y con reglas de negocios muy complejas. Para estas necesidades ya no bastaba la programación estructurada ni mucho menos la programación lineal. Es así como aparece la programación orientada a objetos (POO). La POO viene de la evolución de la programación estructurada; básicamente la POO simplifica la programación con la nueva filosofía y nuevos conceptos que tiene. La POO se basa en la dividir el programa en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les llama objetos. Los objetos son unidades

[2] Ver en el anexo No. 1

independientes que se comunican entre ellos mediante mensajes. A continuación se explica con mayor detenimiento este tema.

Los principios de la POO son: abstracción, encapsulación, modularidad, jerarquía o herencia. Hasta aquí fundamentales y en un grado menos está la tipificación, concurrencia y persistencia.

Ventajas de un lenguaje orientado a objetos:

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos.
- Agiliza el desarrollo de software.
- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.

Para entender este modelo se necesita conocer estos 4 conceptos básicos:

a) Objeto. Existen muchas definiciones de Objeto. ¿Qué es un objeto del mundo real? En realidad son todos los objetos que nos rodean, que se pueden ver y tocar. Estos a la vez poseen dos componentes: características y comportamiento. Por ejemplo los automóviles son objetos que tienen como características (marca, modelo, color, velocidad máxima, etc.) y comportamiento (frenar, acelerar, retroceder, llenar combustible, cambiar llantas, etc.).

Los objetos de software, al igual que los objetos del mundo real, también tienen características y comportamientos. Un objeto de software mantiene sus características en una o más "variables", e implementa su comportamiento con "métodos". Un método es una función o subrutina asociada a un objeto. [3]

[3] Ver en el anexo No. 2

b) Clases. En el mundo real, normalmente se tiene muchos objetos del mismo tipo. Por ejemplo, un teléfono celular de los miles que hay en el mundo. Si se habla en términos de la programación orientada a objetos, podemos decir que el objeto es celular es una instancia de una clase conocida como "celular". Los celulares tienen características (marca, modelo, sistema operativo, pantalla, teclado, etc.) y comportamientos (hacer y recibir llamadas, enviar mensajes multimedia, transmisión de datos, etc.).[4]

Esto mismo se aplica a los objetos de software, se puede tener muchos de un mismo tipo y con las mismas características.

Definición teórica: La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

Definición Teórica: Una instancia es un objeto de una clase en particular.

c) Herencia. La herencia es uno de los conceptos más cruciales en la POO. La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias subclases (la clase que hereda es llamada superclase o clase padre). Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase. De esta manera se crea una jerarquía de herencia.

En general, se puede tener una gran jerarquía de clases tal y como se puede ver en el siguiente gráfico.[5]

d) Envío de Mensajes. Un objeto es inútil si está aislado. El medio empleado para que un objeto interactúe con otro son los mensajes. Hablando en términos un poco más técnicos, los mensajes son invocaciones a los métodos de los objetos.

En resumen se puede decir que:

[4] Ver en el anexo No. 3
[5] Ver en el anexo No. 4

- En un programa orientado a objetos se tendrá a un conjunto de objetos colaborando entre ellos.
- La orientación a objetos es un paradigma del cual se está haciendo uso actualmente para el desarrollo del software.
- Un objeto es una abstracción conceptual del mundo real que se puede traducir a un lenguaje de programación orientado a objetos.
- Un objeto del mundo real tiene características y comportamientos, y de la misma manera, un objeto del mundo del software tiene variables y métodos.
- Una clase es una plantilla que define las variables y métodos a ser incluidas en un tipo de objeto específico.
- Los objetos también son llamados instancias de la clase. Los objetos solo almacenan su estado. De dice que un objeto tiene estado cuando tiene valores en sus variables.
- Los objetos se comunican entre ellos usando los mensajes. Un mensaje es la invocación de un método del objeto.
- La orientación a objetos requiere de una metodología que integre el proceso de desarrollo y un lenguaje de modelación con herramientas y técnicas adecuadas.

1.4 Lenguajes de Programación

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

Aunque muchas veces se usa lenguaje de programación y lenguaje informático como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el HTML.

Un lenguaje de programación permite a un programador especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje léxico.

Los procesadores usados en las computadoras son capaces de entender y actuar según lo indican programas escritos en un lenguaje fijo llamado lenguaje de máquina. Todo programa escrito en otro lenguaje puede ser ejecutado de dos maneras:

- Mediante un programa que va adaptando las instrucciones conforme son encontradas. A este proceso se lo llama interpretar y a los programas que lo hacen se los conoce como intérpretes.
- Traduciendo este programa al programa equivalente escrito en lenguaje de máquina. A ese proceso se lo llama compilar y al traductor se conoce como compilador.

Ejemplos de Lenguajes de Programación:

- FORTRAN
- COBOL
- PL/I
- BASIC
- VISUAL BASIC
- PASCAL
- C
- C++.
- C Sharp.
- LISP
- OBJECT PASCAL
- JAVA
- JAVASCRIPT
- PHP
- ASP
- ADA
- SMALLTALK

1.5 Lenguajes del lado cliente y del lado del servidor

El navegador es una especie de aplicación capaz de interpretar las órdenes recibidas en forma de código HTML, fundamentalmente, y convertirlas en las páginas que son el resultado de dicha orden.[6]

Cuando nosotros se hace clic sobre un enlace hipertexto, en realidad lo que pasa es que se establece una petición de un archivo HTML residente en el servidor (un ordenador que se encuentra continuamente conectado a la red) el cual es enviado e interpretado por nuestro navegador (el cliente).

Así pues, podemos hablar de lenguajes del lado servidor que son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. Por otro lado, los lenguajes del lado cliente (entre los cuales no solo se encuentra el HTML sino también el JavaScript los cuales son simplemente incluidos en el código HTML) son aquellos que pueden ser directamente "digeridos" por el navegador y no necesitan un pre-tratamiento.[7]

1.4.1 Lenguajes del lado cliente

En este proyecto se ha decidido hacer uso como un lenguaje del lado del cliente el JavaScript; este es un lenguaje de programación utilizado para crear pequeños módulos encargados de realizar acciones dentro del ámbito de una página web. Su uso se basa, fundamentalmente, en la creación de efectos especiales en las páginas y la definición de interactividades con el usuario.

JavaScript se representa como un lenguaje de órdenes, fácil de utilizar y capaz de enlazar objetos y recursos propios tanto de HTML como de Java. A diferencia de Java que ha sido diseñado para programadores, JavaScript ha sido pensado para que

[6] Ver en el anexo No. 5
[7] Ver en el anexo No. 6

creadores de páginas web dispongan de una herramienta de sencillo manejo que les permita añadir interactividad a sus trabajos.

La estructura de un documento HTML es:

<HTML>

<HEAD>

<TITLE></TITLE>

</HEAD>

<BODY>

[Contenido de la página]

</BODY>

</HTML>

Las sentencias escritas en JavaScript se encierran entre las etiquetas <SCRIPT LANGUAGE="JavaScript"> </SCRIPT > dentro de las etiquetas </TITLE> y </HEAD>. Es decir que queda de esta manera:

<HTML>

<HEAD>

<TITLE> </TITLE>

<SCRIPT LANGUAGE="JavaScript">

[Aquí debe ir colocado el script]

```
        </SCRIPT >
</HEAD>
<BODY>
        [Contenido de la página]
</BODY>
</HTML>
```

1.4.2 Lenguajes del lado del servidor

El lenguaje C# es un lenguaje de programación desarrollado y estandarizado por Microsoft como parte de su plataforma .Net, que después fue aprobado como un estándar por ECMA e ISO; C Sharp es un lenguaje simple, moderno, seguro y con una fuerte herencia de C/C++".

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (más notablemente de Delphi y Java). C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

Aunque C# forma parte de la plataforma .NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Aunque aún no existen, es posible implementar compiladores que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX.

¿Porque usar C Sharp?

Con la idea de que los programadores más experimentados puedan obtener una visión general del lenguaje, a continuación se recoge de manera resumida las principales características de C#:

- Sencillez: C# elimina muchos elementos de otros lenguajes y que son innecesarios en .NET. Por ejemplo:

  - El código escrito en C# es auto contenido, lo que significa que no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL.

  - El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.

  - No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (.) acceder a miembros de espacios de nombres (::).

- Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular.

- Orientación a objetos: Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos, aunque eso es más bien una característica del CTS que de C#. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

C# soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

- Orientación a componentes: La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).
- Eficiente: En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como inseguras (modificador unsafe) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad procesamiento muy grandes.
- Compatible: Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados Platform Invocation Services (PInvoke), la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32.
- Instrucciones seguras: Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, la guarda de toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de igualdad (==) con el de asignación (=); y todo caso de un switch ha

de terminar en un “break” o “goto” que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.

- Sistema de tipos unificado: A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada System.Object, por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos")
- Extensibilidad de tipos básicos: C# permite definir, a través de estructuras, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos. Es decir, que se puedan almacenar directamente en pila (luego su creación, destrucción y acceso serán más rápidos) y se asignen por valor y no por referencia.
- Extensibilidad de operadores: Para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que C++ y a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores -incluidos los de conversión, tanto para conversiones implícitas como explícitas- cuando se apliquen a diferentes tipos de objetos.
- Extensibilidad de modificadores: C# ofrece, a través del concepto de atributos, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la librería de reflexión de .NET. Esto, que más bien es una característica propia de la plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

En la actualidad existen los siguientes compiladores para el lenguaje C#:

- Microsoft .NET framework SDK incluye un compilador de C#, pero no un IDE.
- Microsoft Visual C#, IDE por excelencia de este lenguaje, versión 2002, 2003 y 2005.
- #develop, es un IDE libre para C# bajo licencia LGPL, muy similar a Microsoft Visual C#.
- Mono, es una implementación GPL de todo el entorno .NET desarrollado por Novell. Como parte de esta implementación se incluye un compilador de C#.
- Delphi 2006, de Borland Software Corporation.
- DotGNU Portable.NET, de la Free Software Foundation.

1.5 Navegadores web

Un navegador web es una aplicación software que permite al usuario recuperar y visualizar documentos de hipertextos comúnmente descritos en HTML, desde servidores web de todo el mundo a través de internet. Los navegadores actuales permiten mostrar o ejecutar: gráficos, secuencias de vídeo, sonido, animaciones y programas diversos además del texto y los hipervínculos o enlaces.

Ejemplos de Navegadores Web:

- Internet Explorer y sus derivados
- Avant Browser
- Maxthon
- G-Browser
- Mozilla y sus derivados
- Mozilla Firefox
- Beonex
- Galeon

- Netscape Navigator
- Opera
- Konqueror
- IBrowse
- AWeb

1.6 Sistemas existentes vinculados al campo de acción

Actualmente se cuenta con el sistema SIH, pero este sistema no cuenta con todos los requerimientos necesarios, está implementado en Visual Basic 6. También existe, el Care2x, es un sistema informático para entidades de salud, desarrollado desde el 2002 por un grupo mundial de programadores y basado en estándares opensource. Care2x integra datos, funciones y flujo de tareas en un entorno de cuidados de la salud. Al momento, está compuesto de cuatro componentes principales. Cada uno de estos componentes también puede funcionar de manera individual, está implementado en PHP.

Se puede concluir que, en este capítulo se hace alusión a las características de las diferentes tecnologías usadas para el desarrollo de la propuesta. Se define el paradigma de programación orientado a objetos y como plataforma de desarrollo .NET, C# como el lenguaje mediante el cual se implementa el negocio. Se definió la arquitectura en tres capas como la que será utilizada. Y finalmente se hizo una investigación acerca de los sistemas existentes relacionados con la gestión hospitalaria.

CAPÍTULO 2: ANALISIS DE LA PROPUESTA

En este capítulo se hace una valoración del diseño propuesto por el analista, se determinan las relaciones de otros módulos y componentes con sistema desarrollado. Además se describen las principales clases que son necesarias para la implementación de la aplicación.

El desarrollo de aplicaciones web involucra decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo, afectando la división de tareas. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de extensión y reusabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a esta, son de vital importancia para el desarrollo de aplicaciones web.

2.1 Valoración crítica del diseño propuesto por el analista

Para la elaboración de un software capaz de satisfacer las necesidades del usuario se ha hecho una propuesta por el analista del sistema; este propone una aplicación web hecha en Asp.Net, lo que trae como una ventaja importante la capacidad de adaptarse sin dificultad a cualquier plataforma ya sea Windows, Linux, etc. Proporciona una interfaz agradable, simple y de fácil uso al usuario. Se ajusta a los estándares establecidos para el desarrollo de un buen diseño. La respuesta de un petición del usuario es rápida, tratando que la aplicación sea los más interactiva posible. Para su uso el usuario le basta con tener conocimientos básicos de informáticos para el trabajo con aplicaciones Windows y ambiente web.

La velocidad de respuesta del sistema depende de la capacidad de respuesta del servidor web por estar diseñado sobre la arquitectura de tres capas.

Para la instalación de la aplicación y posteriormente su uso se requiere un servidor de base de datos PostgreSQL y por parte del cliente un navegador que interprete el HTML independientemente del sistema operativo.

El sistema de seguridad está diseñado para que solo el personal autorizado pueda acceder. Se autenticará mediante un algoritmo de encriptación el MD5, donde la contraseña viajara al servidor encriptada. Durante el proceso de trasferencia de datos se garantiza la seguridad e integridad de los datos realizando en cada momento el trabajo con el servidor y el cliente según corresponda, garantizando que lleguen a su destino y sean los correctos.

Esta propuesta realizada por el analista utiliza patrones de diseño como el patrón de Fabricación Pura, que a su vez reúne el trabajo de los siguientes patrones:

- Baja cohesión
- Alto acoplamiento
- Experto
- Polimorfismo

2.2 Análisis de posibles componentes y módulos existentes que son usados

Para poder dar solución al problema propuesto en el trabajo, es necesario hacer uso de otros módulos que se encuentran implementados como es el caso del módulo de configuración el cual gestiona toda la información referente a los codificadores necesarios para el funcionamiento del cuerpo de guardia. Otro módulo, es el de seguridad el cual gestiona la seguridad y autenticidad de la información de la propuesta.

Se utilizan también, los módulos pertenecientes al mismo proyecto de hospitales que se encuentra en fase de desarrollo, es el caso de inscripción y admisión (cuando al paciente es necesario ingresarlo en el hospital, por el grado de gravedad que tenga), de laboratorio (cuando es enviado el paciente por el médico para realizarse un análisis) así como un módulo aún no implementado que es el de enfermería. En la solución del problema de este trabajo se incluyó la solución del negocio de esos módulos para así poder interactuar con los mismos.

Otro módulo es el middleware, que no es más que una subcapa con la función de acceder al repositorio de datos, está encargada de ejecutar la lógica de acceso a datos, contiene dos paquetes de clases: repositorios y las fábricas de objetos.

Los repositorios son los encargados de manejar las colecciones de objetos comunes (Entidades de la BD representados como objetos) y realizar operaciones sobre ellas.

Fábricas de objetos son los paquetes de clases que contiene la funcionalidad necesaria para acceder a la base de datos y realizar las operaciones que se explican a continuación. Por cada entidad mapeada desde la base de datos, existen cuatro clases que heredan de las interfaces ISelectionFactory: encargada de llevar a cabo el proceso de selección de una entidad determinada en la base de datos, IInsertFactory: encargada del proceso de inserción, IDeleteFactory: Encargada de suprimir una entidad determinada, IUpdateFactory: encargada de actualizar atributos de los objetos en la BD, existe además por cada entidad otra clase que hereda de la clase interfaz IDomainObjectFactory y es la encargada de realizar el proceso de mapeo de las entidades (convertir tupla a tupla el resultado de un proceso de selección en la entidad a la que corresponde).

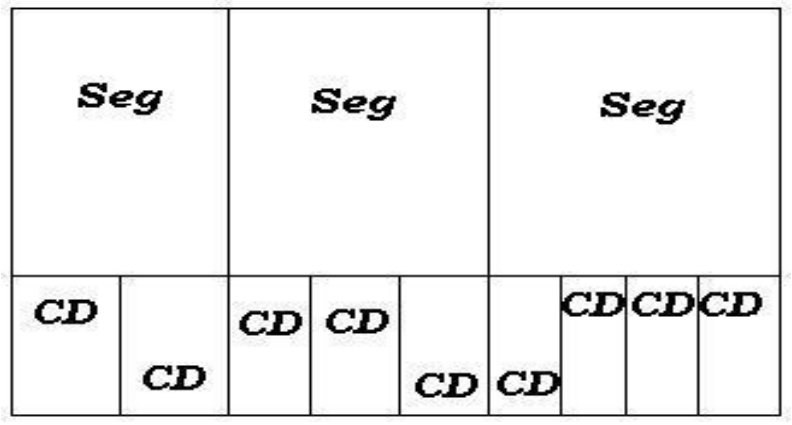
Se hace uso también de NpgSQL que es un abastecedor neto de datos para PostgreSQL este se pone en ejecución en C# y establece una comunicación entre la plataforma .NET y el servidor de bases de datos el PostgreSQL.

El HL7 (Health Level 7) es otro de los módulos del que se hace uso en este trabajo. Este es una especificación para un estándar de intercambio de datos electrónicos en el ambiente de la atención médica en la salud, con un gran énfasis en las comunicaciones dentro de los hospitales. Fue iniciado en el año 1987 a raíz de una conferencia organizada para discutir los problemas de los estándares de la salud a nivel mundial.

El HL7 actualmente se ocupa de las interfaces entre sistemas que emiten o reciben mensajes de registro, admisión, transferencia y alta de pacientes, pedidos de información al sistema, ordenes, resultados, observaciones clínicas, facturación, reportes y actualización de información de archivos maestros. El mismo no asume

ninguna arquitectura en general con respecto a la ubicación de los datos dentro de la aplicación; aunque si está diseñado para dar soporte tanto a un sistema central de atención de pacientes, como a un ambiente más distribuido donde las aplicaciones departamentales son lo repositorios de los datos.

El formato general de los mensajes consiste de campos de datos de longitud variable, separados por caracteres especiales, según reglas específicas de codificación. Los campos de datos se combinan para formar agrupamientos lógicos denominados segmentos, los cuales a su vez están separados entre sí por caracteres específicos. Un mensaje lo componen diversas líneas cada una de las cuales representan líneas. Atento a lo expuesto, un mensaje del HL7 podría ilustrarse de la siguiente manera:



Se hace uso del Chameleon para separar la interfaz de mensaje de las diferentes implementaciones de HL7, lo que permite que una aplicación escrita con una interfaz soporte un gran número de diferentes versiones de HL7. Chameleon permite afrontar los problemas que puede plantear el aislamiento del código de aplicación respecto a la estructura de datos HL7 sin codificar. El Chameleon utiliza el archivo de definición de mensaje (VMD) como interfaz que actúa como puente entre el sistema y el mundo HL7. Es importante aclarar que dicha interfaz se crea dentro del VMD usando tablas, las cuales se usan como objetos[8] con los cuales se interactúa. Los datos son mapeados

[8] Consultar el anexo no. 8

desde los mensajes entrantes y viceversa. Este presenta soporte para los lenguajes más utilizado en la actualidad: C#, Java, Visual Basic, Delphi, etc.

2.2.1 Estrategia de integración entre estos módulos

Toda la estrategia usada en este trabajo para la integración de esto módulos se basa principalmente en realizar llamadas a los métodos o eventos de forma directa. Esta comunicación se realiza mediante llamadas a servicios web o como ya se aborda anteriormente a nivel de negocio, en caso de utilizarse los servicios web se cumplen con los estándares establecidos internacionalmente para facilitar la integración entre nuevos módulos hospitalarios o componentes existentes. Se accede a la base de datos mediante clases controladoras que son las encargadas de controlar a todas aquellas clases interfaces o componentes que se utilicen.

2.3 Estándares de codificación

Los estándares de codificación son reglas específicas a una lengua que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores. Los estándares de codificación no destapan problemas existentes, evitan más bien que los errores ocurran. Con su desarrollo ayudan al programador a producir códigos con alta calidad, al igual que a entender y a utilizar el código por otros programadores.

En los flujos de implementación de este módulo se definió por los arquitectos del proyecto utilizar como estándares de codificación: Mayúsculas y Minúsculas Pascal, Mayúsculas y Minúsculas Camel. También algunos otros parámetros a tomar en cuenta en cuanta a la hora de definir los nombres de las clases, los nombres de las interfaces, el uso de las abreviaturas etc.

El estándar Mayúsculas y Minúsculas Pascal fue diseñado para mantener un solo formato a la hora de escribir líneas de código; este define que la primera letra del identificador y la primera de la siguiente palabra estén en mayúscula.

El otro estándar a usar es el Mayúsculas y Minúsculas Camel, tiene el mismo propósito del otro estándar, pero este lo que define es que va a tener la primera letra del identificador en minúscula y la de letra de la siguiente palabra será escrita en mayúscula.

Uso de los estándares:

- Mayúsculas y minúsculas Pascal
  - Clases
  - Espacios de nombres
  - Propiedades
  - Métodos
- Mayúsculas y minúsculas Camel
  - Atributos
  - Variables utilizadas
  - Campos estáticos y parámetros

Por citar algún ejemplo dentro de este trabajo:

- En la solución se utilizan las clases, Consulta, Recepcion, ExamenFisico, Destino, Código, etc., como se puede apreciar se cumple con lo establecido por los arquitectos del proyecto, ya que los nombres de las clase deben escribirse según el estándar de codificación Mayúsculas y Minúsculas de Pascal.
- En el caso de los nombres de interfaces también se usan aquí basado en el estándar de codificación de Pascal. Ej.: (Gu_Cierre_Guardia, Hc_Agregar_Consulta, Re_Agregar_Consulta, etc.)

- Se utiliza el Mayúsculas y Minúsculas de Camel para escribir los nombres de los parámetros necesarios.
- Los nombres de funciones o métodos fueron escritos mediante el estilo de Pascal.

2.4 Patrón de diseño utilizado

Como parte de una investigación llevada a cabo por lo arquitectos del proyecto se definió utilizar el patrón de diseño: Fabricación Pura, Fábrica Abstracta y Mapeo de Datos.

Se le llama a este patrón de fábrica (factory en inglés) porque involucran algún tipo de factoría o fabrica de objetos. Los objetos de fabricación tienen la responsabilidad de crear instancias de otras clases. Además tienen la responsabilidad y el conocimiento necesario para encapsular la forma que en que se crean determinados tipos de objetos en una aplicación.

Una fábrica es “una clase que implemente uno o más métodos de creación, que son los métodos que se encargan de crear instancias de objetos (estas instancias pueden ser de esta misma clase o de otras). Esta clase tiene entre sus responsabilidades la creación de instancias de objetos, pero puede tener también otras responsabilidades adicionales. Los métodos de creación pueden ser estáticos” [WELICKI, 2007: http://www.microsoft.com/].

Específicamente existen distintos tipos de patrones de fabricación, tales como: simple factory, factory method y abstract factory.

En el presente trabajo se hace uso de estos tres tipos el simple factory por ejemplo es la clase utilizada para crear instancias de otras clases, factory method es que define una interfaz para crear objetos pero deja que sean las subclases las que definan a que clase instanciar; así como el abstract factory es que el proporciona una interfaz para crear familias de objetos relacionados y dependientes entre sí, sin especificar sus clases concretas.

Mediante una aplicación que se creó en el propio proyecto de gestión hospitalaria llamada “Create Factory” por un grupo de estudiantes y asesorado por profesores, se generó la capa intermedia o middleware y las entidades del negocio de este módulo de cuerpo de guardia. Esto se hizo gracias al carácter repetitivo que tiene las mismas.

El patrón Fabricación Abstracta le permite al programador abstraerse, a la hora de implementar su negocio, del soporte de base de datos que se este utilizando.

Fabricación Pura le permite al implementador la forma en que se van a manipular una colección dada de objetos. Este en su estructura posee una clase controladora y una clase repositorio que se encarga de la manipulación de esta colección de objetos mediantes una serie de operaciones básicas de estructuras de datos. A su vez este repositorio depende de clases las cuales tienen implementada la forma en que se manipulará esa colección de objetos. Este patrón soporta alta cohesión puesto que las responsabilidades se factorizan en una clase de grano fino que solo se centra un conjunto de tareas relacionadas.

El Mapeo de Datos se refleja claramente cuando una clase fábrica (factory) es la encargada de mapear a través de una clase de selección (SelectionFactory) una tupla de la base de datos y construir el objeto entidad del negocio.

2.5 Negocio

El negocio comienza desde el momento en que un paciente llega al cuerpo de guardia mediante dos vías: por la vía emergencias o por sí mismo. Este es recibido en la recepción donde se le otorgará un código de urgencias y según como sea considerado pasará a una lista de espera de pacientes los cuales serán atendidos en consulta por el médico de guardia en ese momento; en caso de que sea considerado grave es trasladado urgente a cuidados intensivos donde es ingresado y atendido allí por otro médico.

En el otro caso se le pide los datos personales como nombre, apellidos, número de historia clínica, etc.; esto es para inscribirlo en la lista de espera para la consulta el

médico del cuerpo de guardia. Ya una vez en consulta con el médico de guardia, este le toma los datos para su hoja de cargo, le solicita la historia clínica, en caso posible que este no la posea, se prosigue a buscarla mediante el sistema de inscripción admisión, el cual tiene el registro de todos los pacientes. También el médico es el encargado de realizarle exámenes físicos para así en conjunto con los datos de su historia clínica, que archiva todo su historial médico y los síntomas que presenta el paciente se pueda determinar una enfermedad. En caso de que el paciente se encuentre en la consulta y sea necesaria la realización de un análisis, se procede con el módulo del laboratorio.

Al finalizar el día el jefe de guardia es encargado de realizar todos los informes correspondientes y generar los reportes de la guardia. Definido por los analistas, el negocio va a constar de tres casos de uso significativos, esto son: gestionar consulta, gestionar recepción y gestionar guardia. Más adelante se planteará la vía de solución a este problema mediante la implementación.

2.6 Descripción de las principales clases y operaciones

Para la implementación de este módulo de cuerpo de guardia se necesitó la implementación de tres clases controladoras, las cuales manejarían todo el módulo, ya que ellas estarían basadas en los casos de uso significativos; estas son:

✓ GestionarRecepcion: Para gestionar todo lo relacionado con los pacientes que llegan a la recepción.

```
public class GestionarRecepcion
    {
            public GestionarRecepcion()
            {}


                /// <summary>
                /// Buscar pacientes que hayan sido recepcionados
                /// </summary>
                /// <param name="recepcion">Datos del paciente</param>
```

```
            /// <returns></returns>
        public List<RecepcionView> BuscarPaciente(RecepcionView recepcion)
         {
        RecepcionViewRepositorio repositorio = new
RecepcionViewRepositorio();
        return repositorio.ObtenerTodos(recepcion);
            }

            /// <summary>
            /// Agregar un paciente a la recepcion
            /// </summary>
            /// <param name="recepcion">Datos del paciente</param>
    public void AgregarPaciente(Recepcion recepcion)
            {
         RecepcionRepositorio repositorio = new RecepcionRepositorio();
        repositorio.Adicionar(ref recepcion);
        }

        /// <summary>
            /// Cambiar la Clasificacion de un paciente
        /// </summary>
         /// <param name="recepcion">Datos del paciente</param>
    public void CambiarClasificacion(Recepcion recepcion)
         {
        RecepcionRepositorio repositorio = new RecepcionRepositorio();
        repositorio.Actualizar(recepcion);
         }

         /// <summary>
         /// Enviar un paciente a la consulta del medico
         /// </summary>
         ///    <param    name="recepcion">Datos    del    paciente    de
recepcion</param>
         /// <param name="consulta">Datos de la consulta</param>
    public void EnviarAConsula(Recepcion recepcion, Consulta consulta)
```

```
            {
            RecepcionRepositorio repositorioRecepcion = new
RecepcionRepositorio();
            ConsultaRepositorio repositorioConsulta = new
ConsultaRepositorio();
            repositorioRecepcion.Actualizar(recepcion);
            repositorioConsulta.Adicionar(ref consulta);
                  }

              /// <summary>
              /// Eliminar un paciente de la recepcion
                   /// </summary>
              /// <param name="recepcion">Datos del paciente</param>
        public void EliminarRecepcion(Recepcion recepcion)
            {
            RecepcionRepositorio repositorio = new RecepcionRepositorio();
            repositorio.Actualizar(recepcion);
                  }
 }
```

✓GestionarConsulta: se implementó con vista a manejar lo relacionado con las consultas médicas en los cuerpos de guardia.

```
 public class GestionarConsulta
     {
        public GestionarConsulta()
        {}

private      List<RegistroPacienteView>      ListaDatosPersona      =      new
List<RegistroPacienteView>();

              /// <summary>
              /// Buscar consultas de pacientes
              /// </summary>
```

```
            /// <param name="consulta">Datos de la consulta</param>
                 /// <returns></returns>
        public List<ConsultaView> BuscarConsulta(ConsultaView consulta)
              {
            ConsultaViewRepositorio repositorio = new
ConsultaViewRepositorio();
            return repositorio.ObtenerTodos(consulta);
             }

             /// <summary>
             /// Agrega los pacientes a la consulta
             /// </summary>
             /// <param name="consulta">Datos de consultas</param>
public void AgregarConsulta(Consulta consulta, Recepcion recepcion,
List<CGExamenFisico> listaExamenes)
               {
             ConsultaRepositorio repositorioConsulta = new
ConsultaRepositorio();
                  RecepcionRepositorio repositorioRecepcion = new
RecepcionRepositorio();
        CGExamenFisicoRepositorio repositorioExamen = new
CGExamenFisicoRepositorio();
            repositorioRecepcion.Adicionar(ref recepcion);
            consulta.IdRecepcion = recepcion.IdRecepcion;
            repositorioConsulta.Adicionar(ref consulta);
                  for (int i = 0; i < listaExamenes.Count; i++)
                       {
                        CGExamenFisico temp = new CGExamenFisico();
                        listaExamenes[i].IdConsulta = consulta.IdConsulta;
                        temp = listaExamenes[i];
                        repositorioExamen.Adicionar(ref temp);
                      }
              }

              /// <summary>
```

```
            /// Edita los datos de una consulta
            /// </summary>
            /// <param name="consulta">Datos de la consulta</param>
 public    void    EditarConsulta(Consulta    consulta,    List<CGExamenFisico>
listaExamenes)
            {
             CGExamenFisicoRepositorio      repositorioExamen      =      new
CGExamenFisicoRepositorio();
                ConsultaRepositorio repositorio = new ConsultaRepositorio();

                  repositorio.Actualizar(consulta);

                 for (int i = 0; i < listaExamenes.Count; i++)
                   {
                   CGExamenFisico temp = new CGExamenFisico();
                   listaExamenes[i].IdConsulta = consulta.IdConsulta;
                   temp = listaExamenes[i];
                   repositorioExamen.Adicionar(ref temp);
                   }

            }

                /// <summary>
           /// Elimina una consulta
           /// </summary>
           /// <param name="consulta">Datos de la consulta</param>
        public void EliminarConsulta(Consulta consulta)
        {
            ConsultaRepositorio repositorio = new ConsultaRepositorio();
            repositorio.Eliminar(consulta);
        }

        /// <summary>
        /// Remitir Paciente para otra especialidad
        /// </summary>
```

```
        /// <param name="consulta">Datos de la consulta</param>
        public void RemitirConsultaOtraEspecialidad(Consulta consulta)
        {
            ConsultaRepositorio repositorio = new ConsultaRepositorio();
            repositorio.Actualizar(consulta);
        }

        /// <summary>
        /// Enviar a la sala de observaciones del cuerpo de guardia
        /// </summary>
        /// <param name="consulta">Datos de la consulta</param>
        public void EnviarObservacion(Consulta consulta)
        {
            ConsultaRepositorio repositorio = new ConsultaRepositorio();
            repositorio.Actualizar(consulta);
        }
    }
```

También se hizo necesario la implementación de dos clases de `ListaRecepcion` y `ListaConsulta`, con el objetivo de poder crear listas de objetos con todos los atributos necesarios de cada una de ellas para el trabajo con las mismas.

2.6.1 Gestión de Consulta

Nombre: CC_Gestionar_Consulta	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Eliminar_Consulta(consulta)
Descripción:	Elimina una consulta dada, siempre que sea personal autorizado para ello.
Nombre	Agregar_Consulta(consulta)
Descripción:	Agrega una nueva consulta realizada por el

	medico.
Nombre:	Buscar_Consulta(consulta)
Descripción:	Se busca una consulta en caso de que sea necesario.

Nombre: CE_Consulta	
Tipo de clase: Entidad	
Atributo	**Tipo**
Id_Consulta	Interger
Id_Recepcion	Interger
Id_Tipo_consulta_cuerpo_guardia	Interger
Id_Conducta	Interger
Id_Destino	Interger
Id_Paciente	Interger
Id_Diagnostico	Interger
Id_Funcionario	Interger
Fecha_hora_atencion	Date Time
Fecha_hora_salida	Date Time
Tratamiento	Varchar
Sintomas	Varchar
Observaciones	Varchar
Para cada responsabilidad.	
Nombre	**Descripción**
get Id_Consulta	Devuelve el identificador de una consulta.
get Id_Recepcion	Devuelve el identificador de un

	caso que llega a la recepción del cuerpo de guardia.
get Id_Tipo_consulta_cuerpo_guardia	Devuelve el identificador de una consulta determinada en el cuerpo de guardia.
get Id_Conducta	Devuelve el identificador de una conducta a seguir.
get Id_Destino	Devuelve el identificador del destino.
get Id_Paciente	Devuelve el identificador de un paciente.
get Id_Diagnostico	Devuelve el identificador de un diagnóstico.
get Id_Funcionario	Devuelve el identificador de un funcionario.
get Fecha_hora_atencion	Devuelve la hora en que fue atendido el paciente.
get Fecha_hora_salida	Devuelve la hora en que el paciente salió de la consulta.
get Tratamiento	Devuelve el tratamiento a seguir por el paciente.
get Sintomas	Devuelve los síntomas que presenta el paciente.
get Observaciones	Devuelve las observaciones hechas por el medico de la consulta.
set Id_Consulta	Establece el identificador de una consulta determinada.

set Id_Recepcion	Establece el identificador de un caso que llega a la recepción de un cuerpo de guardia.
set Id_Tipo_consulta_cuerpo_guardia	Establece el identificador de un tipo de consulta del cuerpo de guardia.
set Id_Conducta	Establece el identificador a un una conducta a seguir.
set Id_Destino	Establece el identificador del destino del paciente.
set Id_Paciente	Establece el identificador del paciente.
set Id_Diagnostico	
set Id_Funcionario	Establece el identificador de un funcionario.
set Fecha_hora_atencion	Establece la hora en que fue atendido un paciente.
set Fecha_hora_salida	Establece la hora en que se retiró de la consulta el paciente.
set Tratamiento	Establece el tratamiento.
set Sintomas	Establece los síntomas.
set Observaciones	Establece las observaciones del médico.

2.6.2 Gestión Examen Físico

Nombre: CE_Examen_fisico

Tipo de clase: Entidad	
Atributo	**Tipo**
Id_Consulta	Interger
Fecha_hora	Date Time
Peso	Double
Temperatura	Double
Frecuencia_cardiaca	Double
Frecuencia_respiratoria	Double
Presion_arterial_minima	Double
Presion_arterial_maxima	Double
Para cada responsabilidad.	
Nombre	**Descripción**
get Id_Consulta	Devuelve el identificador de una consulta.
get Fecha_hora	Devuelve la fecha y hora del examen físico.
get Peso	Devuelve el peso del paciente.
get Temperatura	Devuelve la temperatura que presentó el paciente.
get Frecuencia_cardiaca	Devuelve la frecuencia cardíaca del paciente.
get Frecuencia_respiratoria	Devuelve la frecuencia respiratoria.
get Presion_arterial_minima	Devuelve la presión arterial mínima.
get Presion_arterial_maxima	Devuelve la presión arterial máxima.
set Id_Consulta	Establece el identificador de consulta
set Fecha_hora	Establece la fecha y hora en que fue examinado el paciente.
set Peso	Establece el peso del paciente.

set Temperatura	Establece la temperatura del paciente.
set Frecuencia_cardiaca	Establece la frecuencia cardíaca.
set Frecuencia_respiratoria	Establece la frecuencia respiratoria.
set Presion_arterial_minima	Establece la presión arterial mínima.
set Presion_arterial_maxima	Establece la presión arterial máxima.

2.6.3 Gestión en la Recepción

Nombre: CC_Gestionar_Recepción	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	clasificarPaciente(recepcion)
Descripción:	Posibilita clasificar el paciente según el estado físico en que llegue al cuerpo de guardia.
Nombre:	cambiarClasificacion(recepcion)
Descripción:	Permite modificar el estado físico del paciente.
Nombre:	enviarAConsulta(recepcion)
Descripción:	Permite enviar un paciente que llega a la recepción a una consulta con un médico.
Nombre:	eliminarPaciente(recepcion)
Descripción:	Permite eliminar un paciente de la recepción.
Nombre:	buscarPacienteRecepcion(recepcion)
Descripción:	Busca un paciente en la recepción.

Nombre: CE_Recepción
Tipo de clase: Entidad

Atributo	**Tipo**
Id_Recepcion	Serial
Id_Estado_paciente	Integer
Id_Codigo	Varchar
Id_Tipo_llegada	Integer
Id_Procedencia	Integer
Id_Paciente	Integer
Id_Funcionario	Integer
Fecha_hora_llegada	DateTime

Para cada responsabilidad:	
get Id_Recepcion	Devuelve el identificador de la clase entidad CE_Recepcion.
get Id_estado_paciente	Devuelve el identificador de estado_paciente.
get Id_codigo	Devuelve el identificador de código.
get Id_tipo_llegada	Devuelve el identificador de tipo_llegada, quiere decir mediante que vía llega el paciente; esta puede ser por sus propios medios o fue llevado por el SIUM.
get Id_procedencia	Devuelve el identificador de la tabla de procedencia del paciente.
get Id_Paciente	Devuelve el identificador la tabla de paciente.
get Id_Funcionario	Devuelve el identificador del funcionario que atiende al paciente en la recepción.
get Fecha_hora_llegada	Devuelve la fecha de llegada del paciente a la recepción.
set Id_Recepcion	Establece el identificador de la recepción.
set	Establece el identificador del estado del paciente.

Id_Estado_paciente	
set Id_Codigo	Establece el identificador del código de gravedad del paciente.
set Id_Tipo_llegada	Establece el identificador del tipo de llegada.
set Id_Procedencia	Establece el identificador de l procedencia.
set Id_Paciente	Establece el identificador del paciente
set Id_Funcionario	Establece el identificador del funcionario.
set Fecha_hora_llegada	Establece la fecha y hora de llegada del paciente.

2.6.4 Gestión del Cierre de Guardia

Nombre: CC_Gestionar_Cierre_Guardia	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Cerrar_Guardia(guardia)
Descripción:	Permite hacer un cierre de guardia, dejando almacenado el nombre del médico que hizo el cierre así como la hora en que se hizo. Aquí es donde posteriormente se puede generar los reportes del resumen de la misma.

Nombre: CE_Cerrar_Guardia	
Tipo de clase: Entidad	
Atributo	**Tipo**
Id_Funcionario	Integer
Ultima_fecha_cerrada	DateTime

Para cada responsabilidad:	
Nombre	**Descripción**
get Id_Funcionario	Devuelve el identificador del funcionario.
get Ultima_fecha_cerrada	Devuelve la fecha de la última guardia cerrada.
set Id_Funcionario	Establece el identificador del funcionario.
set Ultima_fecha_cerrada	Establece la fecha de la última guardia cerrada.

2.6.5 Gestión de remisiones

Nombre: CC_Gestionar_remision	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Agregar_remision(remision)
Descripción:	Permite remitir un paciente mediante el sistema.
Nombre:	seleccionar_remision(remision)
Descripción:	Permite la selección de cualquier remisión que se halla hecho con anterioridad y por supuesto su información.

Nombre: CE_Remision	
Tipo de clase: Entidad	
Atributo	**Tipo**
Id_remision	Integer
Id_consulta	Integer

Id_paciente	Integer
Id_especialidad	Integer
Id_funcionario	Integer
fecha_hora_remision	DateTime
descripcion	VarChar
Para cada responsabilidad:	
get Id_remision	Devuelve el identificador de la remisión.
get Id_consulta	Devuelve el identificador de la consulta.
get Id_paciente	Devuelve el identificador del paciente.
get Id_especialidad	Devuelve el identificador de la especialidad de donde ha sido remitido el paciente.
get Id_funcionario	Devuelve el identificador del funcionario que lo remitió.
get fecha_hora_remision	Devuelve la hora y fecha de la remisión.
get descripcion	Devuelve la descripción.
set Id_remision	Establece el identificador de la remisión.
set Id_consulta	Establece el identificador de la consulta.
set Id_paciente	Establece el identificador del paciente.
set Id_especialidad	Establece el identificador de la especialidad.
set Id_funcionario	Establece el identificador del funcionario.
set fecha_hora_remision	Establece la fecha y hora de la remisión.
set descripcion	Establece una breve descripción acerca del paciente.

En este capítulo se hizo un estudio crítico del diseño propuesto por el analista. Se analizaron los patrones de diseño utilizados para la implementación, además de los estándares de codificación usados. Se planteó la estrategia de integración relacionadas con los módulos que interactúan con el de cuerpo de guardia; y se concluyó con una descripción de las principales clases necesarias para el desarrollo de la aplicación.

CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación, es la fase de pruebas. Esta fase del desarrollo de un software es la que mayor cantidad de tiempo y de esfuerzo requiere, se estima que la mitad del esfuerzo de desarrollo de un programa tanto en tiempo como en gastos se invierte en la misma.
En ella se le añade valor al producto, todos los programas poseen errores y la fase de pruebas los descubre, siendo este el valor que le añade. Mientras más errores se encuentren en el software es mejor. Una prueba del software es un conjunto de actividades que se llevan a cabo sistemáticamente, que pueden planificarse por adelantado y ejecutarse una vez construido el código para la revisión final de las especificaciones, del diseño y de la codificación del software.

3.1 Prueba de unidades

Hay multitudes de conceptos asociados a la fase de prueba de un software, existen pruebas de unidades de Caja Blanca y de Caja Negra. Todas estas pruebas de unidades se plantean a pequeña escala, y consisten en ir probando uno a uno los diferentes módulos que constituyen una aplicación. Básicamente la fase de prueba de unidades comienza desde que se compila y se ejecuta el programa; si este presenta errores ahí mismo se repara.

Existen también pruebas de integración y de aceptación pero estas son pruebas a mayor escala que pueden llegar a dimensiones industriales cuando el número de módulos es elevado, o la funcionalidad que se espera del programa es muy compleja.

Las pruebas de integración se centran en probar la coherencia semántica entre los diferentes módulos, tanto de semántica estática como de semántica dinámica; normalmente estas pruebas se vienen efectuando por etapas, abarcando progresivamente más módulos en cada etapa.
Las pruebas de aceptación son las pruebas que decida el cliente que se le deba aplicar al producto antes de darlo por bueno y pagarlo.

Pero como se planteó anteriormente, no se utilizará las pruebas de integración ya que estas son para probar un número grande de módulos, ni las pruebas de aceptación ya que este producto no tiene un cliente que exija hacerle pruebas.

En este trabajo se aplicarán técnicas de Caja Blanca, estas se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar.[9]

3.2 Descripción de las pruebas de unidades a utilizar

3.2.1 Caja blanca

La técnica de la Caja Blanca consiste en realizar pruebas para verificar que líneas específicas de código funcionan tal como está definido. También se le conoce como prueba de caja transparente. Esta se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar.

Estas tienen como objetivos:

- Garantizar que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejercitar todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecutar todos los bucles en sus límites operacionales.
- Ejercitar las estructuras internas de datos para asegurar su validez.

[9] Ver en el anexo No. 7

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad.

Existen varias técnicas para la aplicación de las pruebas de caja blanca, en este trabajo se utiliza la técnica del camino básico. Los pasos que se siguen para aplicar esta técnica son:

1. Se numeran las instrucciones del algoritmo y se construye el grafo de flujo. Las instrucciones simples se agrupan en un mismo nodo. Las condiciones compuestas se separan en varios nodos.
2. Calcular la complejidad ciclomática del grafo. Esta indica el número máximo de caminos linealmente independientes dentro del grafo.

V(G) = Nº de regiones del grafo

V(G) = Nº de Aristas - Nº de Nodos + 2

V(G) = Nºde Nodos Predicado + 1

3. Diseñar los caminos linealmente independientes de menor a mayor de manera que exista a lo máximo una arista de diferencia entre ellos.
4. Elaborar los casos de prueba (datos de entrada al algoritmo y resultados esperados) para cada uno de los caminos.

3.2.2 Caja negra

La prueba de caja negra se centra, principalmente, en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca.

Estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa si lo que se desea es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

Existen varias técnicas, entre ellas están:

1. Técnica de la partición de equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
2. Técnica del análisis de valores límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
3. Técnica de grafos de causa-efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Dentro del método de caja negra la técnica de la partición de equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas

existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico.

La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar. En este apartado se hará uso de esta técnica para aplicarle a cierto caso de uso del sistema desarrollado el método de la caja negra.

3.3 Desarrollo de la prueba aplicando la técnica de la caja blanca

Cómo se hace uso de la prueba de caja blanca con la utilización de la técnica del camino básico, se encaminará este apartado a aplicar los pasos necesarios para la aplicación de esta técnica. A continuación se presenta un segmento de código extraído de una función del algoritmo, la cual realiza una búsqueda de los pacientes que se encuentran en la recepción y los muestra en un listado.

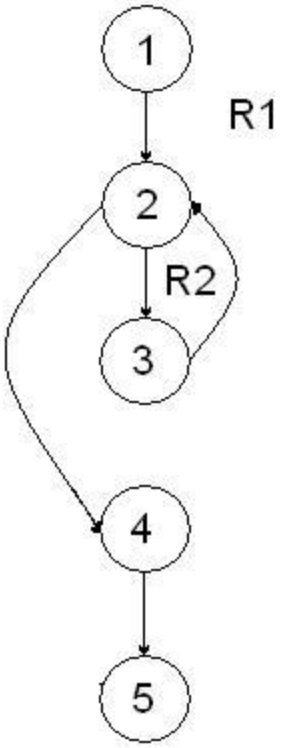
Primeramente, se escogió un procedimiento, el cual no devuelve nada, tiene como función agregar una nueva consulta hecha por el médico al paciente, la cual puede tener uno o varios exámenes físicos realizados.

```
  public void AgregarConsulta(Consulta consulta, Recepcion recepcion,
List<CGExamenFisico>  listaExamenes)
        1 {
        1   ConsultaRepositorio repositorioConsulta = new ConsultaRepositorio();
        1  RecepcionRepositorio repositorioRecepcion = new RecepcionRepositorio();
    1 CGExamenFisicoRepositorio repositorioExamen = new GExamenFisicoRepositorio();
        1  repositorioRecepcion.Adicionar(ref recepcion);
        1  consulta.IdRecepcion = recepcion.IdRecepcion;
        1  repositorioConsulta.Adicionar(ref consulta);
          2   for (int i = 0; i < listaExamenes.Count; i++)
              {
```

```
        3 CGExamenFisico temp = new CGExamenFisico();
       3  listaExamenes[i].IdConsulta = consulta.IdConsulta;
       3 temp = listaExamenes[i];
       3 repositorioExamen.Adicionar(ref temp);
     }4
}5
```

3.3.1 Grafo de flujo

Construyendo el grafo de flujo:



3.3.2 Complejidad ciclomática del grafo

1era vía

V(G) = Nº de regiones del grafo

V(G) = 2

2do vía

V(G) = Nº de Aristas - Nº de Nodos + 2

V(G) = 5 – 5 + 2

V(G) = 2

3era vía

V(G) = Nºde Nodos Predicado + 1

V(G) = 1 + 1

V(G) = 2

El dato V(G) es la complejidad ciclomática del grafo de flujo, la misma define el número de caminos básicos por los cuales se recorre el algoritmo y de esta forma encontrar algún error en el funcionamiento de este.

3.3.3 Hallar los caminos independientes

Camino #1: 1, 2, 4, 5

Un objeto de tipo consulta

Un objeto de tipo recepción

Una lista de exámenes físicos

Resultado Esperado: Asumiendo que el listado de exámenes físicos que se le pasa inicialmente al procedimiento estaba vacio entonces, a la hora de compilarse se llega al ciclo y se encuentra que listaExamenes.Count = 0 por lo tanto no entra y pasa al final del procedimiento, agregando una consulta sin exámenes físicos.

Camino #2: 1, 2, 3, 2, 4, 5
Un objeto de tipo consulta
Un objeto de tipo recepción
Una lista de exámenes físicos
Resultado Esperado: En este caso de prueba se asumió que la lista si tenía exámenes realizados. Por lo que se ejecuta mientras que i < listaExamenes.Count, entonces se agrega una consulta con sus exámenes correspondientes.

Se puede concluir que esta prueba ha demostrado que los caminos lógicos del software, estructura lineal y la implementación del software están correctamente. Teniendo en cuenta que el algoritmo escogido fue el de mayor grado de dificultad y la prueba de la caja blanca resultó satisfactoria porque los resultados fueron los esperados, entonces se puede afirmar que el código no posee ningún error. Se recomienda aplicar estas pruebas de forma iterada durante su desarrollo como mínimo cinco veces para que así se considere el software con mayor calidad.

3.4 Desarrollo de la prueba de la caja negra
Proceso basado en el caso de uso gestionar recepción:

Clases válidas	Clases inválidas	Resultados de la prueba	Resultado de la prueba	Observaciones
El recepcionista del cuerpo de guardia observa un listado con los pacientes que se encuentran en espera a ser	Que aparezcan pacientes que estén ya en consulta.	Se observa la información indicada.	Satisfactorio	Como el listado está ordenado según el estado de

consultados.				salud del paciente.
Se selecciona un paciente del listado para poder hacerle alguna acción indicada en el menú. En la figura que existe para seleccionarlo.	Al seleccionar el paciente los botones del menú no se active.	Se selecciona el paciente deseado y se activan los botones.	Satisfactorio	
Se selecciona entre las opciones que tiene el recepcionista a realizarle al paciente la de enviarlo a consulta. Se hace clic en el botón *Enviar a Consulta*.	No se abra la ventana donde aparecen los médicos de guardia, los cuales pueden atender al paciente.	Se abre la ventana con la información correcta. Funciona el botón.	Satisfactorio	
El recepcionista selecciona el medico indicado	Al seleccionar al médico y	Si se envía el paciente a la	Satisfactorio	Como el listado de los

para enviarle el paciente a la consulta y oprime el botón *Aceptar*.	oprimir el botón aceptar no se envíe el paciente a la consulta.	consulta del medico deseado y desaparece del listado de los pacientes en recepción.		pacientes en recepción se actualiza.
En caso que el recepcionista no desee enviar al paciente todavía a la consulta por algún motivo, oprime el botón *Cancelar*.		Se cierra la ventana sin provocar ningún cambio.	Satisfactorio	
En la página inicial del recepcionista de nuevo. Este desea cambiarle la clasificación a un paciente ya que este se agravó. Oprime *Cambiar Clasificación*.	No funciona el botón.	Si funciona el botón y con el se abre la ventana esperada para cambiarle la clasificación.	Satisfactorio	

Aparecen todas las clasificaciones que se le puede otorgar al paciente en un menú, el recepcionista selecciona la deseada y posteriormente oprime el botón de A*ceptar* para cambiarle esta clasificación.	No se cambie la clasificación.	Se le cambia la clasificación.	Satisfactorio	El listado aparece actualizado y otra vez se ordena por el orden de gravedad de los pacientes ya una vez hecho el cambia que deseara el recepcionista .
En caso de no querer realizar esta operación se oprime *Cancelar.*	No se cierra la página de cambiar clasificación .	Se cancela la operación.	Satisfactorio	Todo continúa como antes, sin producir cambios según lo deseado.
Una vez más el	Al presionar	Se elimina el	Satisfactorio	

recepcionista en su página inicial. Tiene también la opción de eliminar a un paciente de la recepción, para eso está el botón *Eliminar Recepción*	el botón no se elimine la persona de la recepción.	paciente de la recepción.		
La aplicación también posee un menú al cual el recepcionista puede agregar un paciente a la lista de espera en la recepción del cuerpo de guardia. Para esto existe un hipervínculo que dice *Agregar Paciente.* Al hacer clic en él debe abrirse otra página.	Que no funcione el hipervínculo.	Sirvió el hipervínculo.	Satisfactorio	
En la página de		Se agrega el	Satisfactorio	

agregar el paciente a la recepción se llenan los datos correspondientes. Código: Rojo. Forma llegada: ambulancia. Procedencia: Su casa. Los demás datos pertenecen a otro módulo IA que serian los datos de las personas, como Nro. de historia clínica, nombre etc.		paciente a la lista de espera en la recepción.		
También en el menú del recepcionista puede buscar un paciente que haya estado en recepción. Se llenan los datos correspondiente		Se busca todos los pacientes que con ese nombre han pasado por la	Satisfactorio.	

s a la búsqueda deseada. Se puede buscar por nombre, por historia clínica, por la clasificación, etc. Se introduce un Nombre: Juan		recepción.		
Se introduce: Nro. HC:542178452		Se busca los datos de la persona con ese Nro. de historia clínica que se introdujo.	Satisfactorio.	
Se escoge: Código: Rojo		Se busca todos los pacientes que hayan llegado con código rojo	Satisfactorio.	

		a la recepción.		
Se introducen dos fechas: Entre: 22/06/2007 Y: 30/06/2007		Se busca todos los datos de las personas que acudieron entre esas fechas a la recepción del cuerpo de guardia.	Satisfactorio.	
El recepcionista puede solicitar el servicio de enfermería. En el menú hay un hipervínculo el cual realiza esa función.		Se solicita el servicio de enfermería.	Satisfactorio.	

Se puede concluir esta fase de prueba con la aplicación de la técnica de caja negra que el análisis al funcionamiento de los requisitos funcionales del caso de uso de gestionar recepción, están en perfecto estado. Y el diseño propuesto por los analistas y diseñadores del sistema se ajusta afinadamente a las necesidades presentadas en los cuerpos de guardia de los hospitales.

CONCLUSIONES

Con el desarrollo de este trabajo se arribó a las siguientes conclusiones:

- Se implementó una aplicación para la informatización de la atención médica de los pacientes en los cuerpos de guardia.
- En la implementación de dicha aplicación se utilizaron los patrones de diseño establecidos en el análisis.
- Se llevó a cabo un sondeo acerca de la utilización de los servicios que brinda una aplicación web.
- También se indagó acerca de todos los servicios vinculados con la gestión de la atención médica de los pacientes que acuden a un cuerpo de guardia de un hospital.

De esta forma se le da cumplimiento a los objetivos trazados para la elaboración del trabajo diploma obteniendo la implementación de un sistema informático que permita mejorar los procesos de gestión de la atención médica en los cuerpos de guardia en los hospitales.

RECOMENDACIONES

- Implementar próximas versiones en las que se pueda generar e imprimir reportes.
- Realizar un manual de ayuda para un mejor trabajo con el sistema.
- Desplegar este sistema en todos los hospitales y centros hospitalarios del país.

BIBLIOGRAFÍA

- ARCHER, TOM: A Fondo C #.Edit. Mcgraw-Hill / Interamericana De España, S.A, 2001.

- BOOCH, G: "Object Oriented Development", en Trans. of Soft. Eng. Vol. Se-12, Num. 2, 1986, p. 211-221.

- BRUEGGE, BERND, DUTOIT, ALLEN: Ingeniería de software: una perspectiva orientado a objetos, Edit. Pearson Educación, 2002.

- CHARTE OJEDA, FRANCISCO: Visual C# .Net. Edit. Anaya Multimedia-Anaya Interactiva, 2006.

- GÓMEZ MONT, CARMEN: "Nuevas tecnologias de comunicación". Edit. Trillas, 1991.

- GREIFF. W. R.: "Paradigma vs Metodología", El Caso de la POO (Parte II). Edit. Soluciones Avanzadas. 1994, p. 31-39.

- JACOBSON, IVAR: RUMBAUGH, JAMES, BOOCH, GRADY: "El lenguaje unificado de modelado". Edit Addison-Wesley, 2000. (Cap. 11 p 281-302, cap. 16 p. 381-394).

- LATORILLA, ELPIDIO: Care2x. Sistema Información Hospitalario, en http://care2x.ourproject.org, 3 de mayo del 2007.

- LANE, RECHTIN, BHANSALI, GARLAN, PERRY, CREMEN, et al. 2003. Published Software Architecture Definitions, Bibliographic Definitions [Electronic Version] extraído de http://www.sei.cmu.edu/architecture/published_definitions.html#Bibliographic. El 30 de mayo del 2007.

- MACDONALD, MATTHEW: Asp.Net Manual de Referencia. Edit. Mcgraw-Hill 2002.

- MINSAP: Ministerios de salud publica, en http://www.dne.sld.cu/minsap/ Extraído el 30 de abril del 2007.

- MRIDULA, PARIHAR: La Biblia De Asp.Net. Edit Anaya Multimedia-Anaya Interactiva, 2002.

- PRESSMAN,ROGER: "Ingeniería del Software. Un enfoque práctico", Edit. McGraw-Hill/Interamericana de España, 2002.

- PUSSACQ Laborde; Juan Pablo: "Condiciones y casos de prueba", en http://www.rmya.com.ar, 2003. Extraído el 27 de mayo del 2007.

- REYNOSO, C. Y KICILLOF, N.: Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, publicado en el año 2004, última actualización: 05/04/2007. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#24.

- RECIO, FRANCISCO Y PROVENCIO, DAVID: Arquitectura básica de la plataforma .Net. Descripción del Framework y sus principales componentes: Lenguajes, biblioteca de clases y CLR. En http://www.desarrolloweb.com/articulos/1328.php. Extraído el 20 de marzo del 2007.

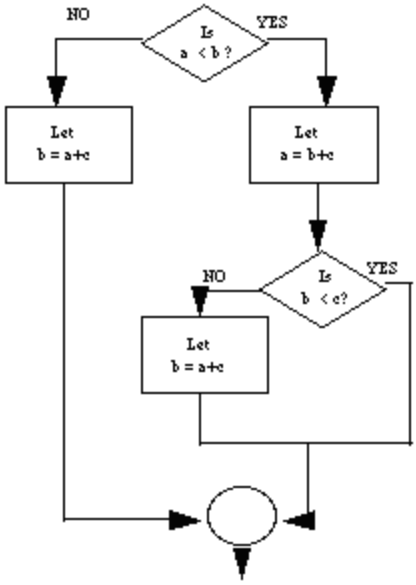
- WELICKI: Patrones de Fabricación: Fábricas de Objetos, en http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3624.asp, 2007.

- ZABALA, R: Ingeniería de Softwareen, en http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html. Extraído el 22 de junio del 2007.

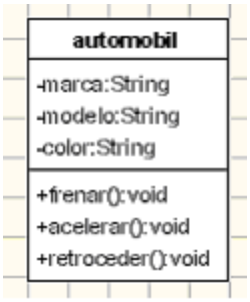
- ZELDMAN, JEFFREY: "Diseño con estándares Web". Edit. Anaya Multimedia-Anaya Interactiva, 2004.

ANEXOS

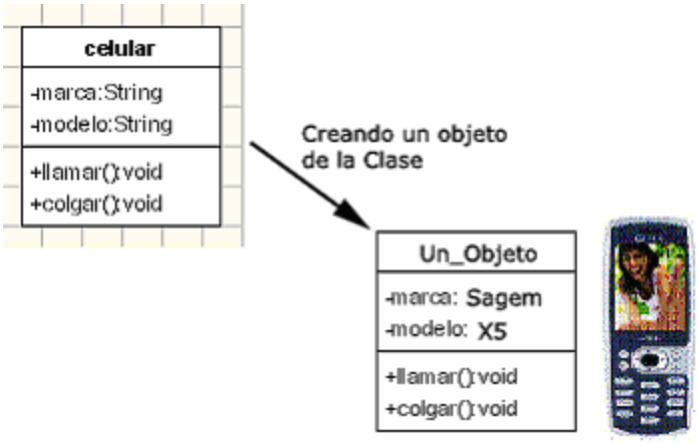
1. Programacion lineal:



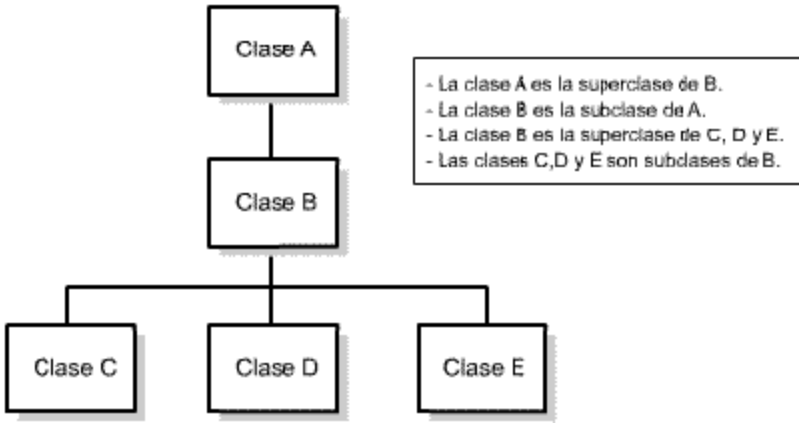
2. Objeto:



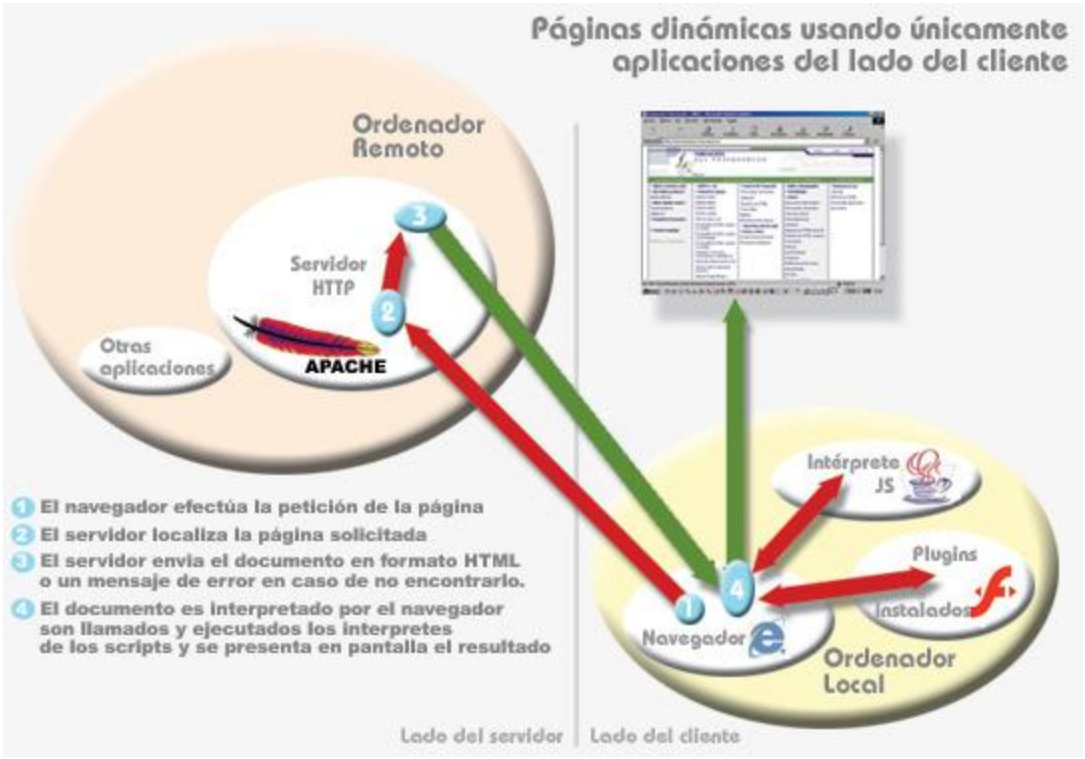
3. Clase:



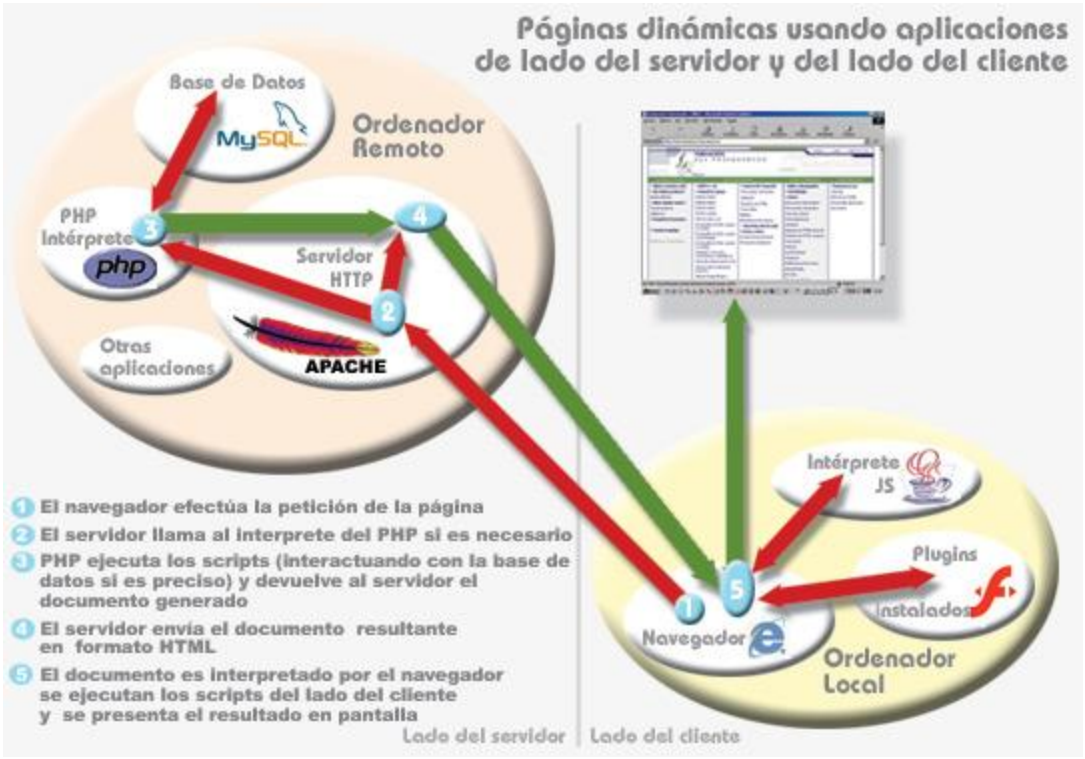
4. Herencia:



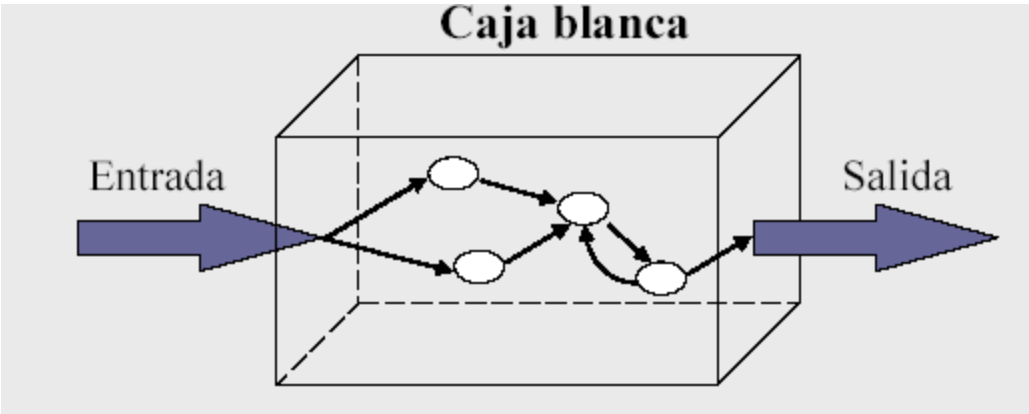
5. Páginas dinámicas usando aplicaciones del lado del cliente:



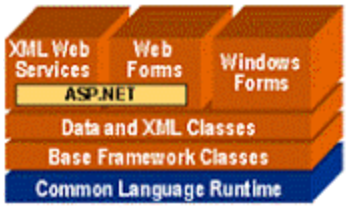
6. Páginas dinámicas usando aplicaciones del lado del servidor y del cliente:



7. Técnica de la caja blanca:



8. CLR (Common Language Runtime):



GLOSARIO

Compiladores: Los compiladores “son programas o herramientas encargadas de compilar. Un compilador toma un texto escrito en un lenguaje de alto nivel (código fuente) y los traduce a un lenguaje comprensible por las computadoras (código objeto)” [ALEGSA, 2007: http://www.alegsa.com.ar/Dic/compilador.php].

Complejidad ciclomática: es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Nodo: Cada círculo representado se denomina nodo del grafo de flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión.

Aristas: Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

Regiones: Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran y la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

Internet: es un método de interconexión entre redes de computadoras, implementado en un conjunto de protocolos TCP/IP y garantiza que redes heterogéneas funcionen como una red única, de ahí es de donde surge el termino “red de redes”.

Http (hipetext transfer protocol): es un protocolo utilizado durante cada transacción de la Web, este protocolo de trasferencia es el sistema mediante el cual se envían las peticiones de acceso a una página. También sirve para enviar información adicional en ambos sentidos, como formularios en campos de texto.