Universidad de las Ciencias Informáticas

Facultad 7

Título: *Implementación del Sistema de Gestión de Información de los Pacientes en los Consultorios Médicos*

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Dailyn Peña Piña

Reinier Hernández Rodríguez

Tutor: *Ing. Yeinier Ferrás Cecilio*

Ciudad de la Habana, Julio del 2008.

"Año 50 de la Revolución"

PENSAMIENTO

Hagamos el propósito de redoblar nuestros esfuerzos y juremos ante nosotros mismos que si un día nuestro trabajo nos pareciera bueno, debemos luchar por hacerlo mejor, y si fuera mejor, debemos luchar por hacerlo perfecto, conociendo de antemano que para un comunista nada será nunca suficientemente bueno, y ninguna obra humana será jamás suficientemente perfecta.

Fidel Castro Ruz

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 firmamos la presente a los 3 días del mes de Julio del año 2008.

Firma del Autor	Firma del Autor	Firma del Tutor
Dailyn Peña Piña	Reinier Hernández Rodríguez	Yeinier Ferrás Cecilio

DATOS DE CONTACTO

TUTOR: Ing. Yeinier Ferrás Cecilio. Profesor graduado como Ingeniero en las Ciencias Informáticas en la Universidad de Las Ciencias Informáticas en el año 2007.Ha impartido la asignatura de Física.
Email: yferras@uci.cu.

ASESOR: Lic. César Nicolás Richard Martínez, Vicedecano de Formación en la Facultad 7.
Email: crichard@uci.cu.

AGRADECIMIENTOS

A nuestro eterno Comandante en Jefe Fidel Castro Ruz y a todos los que hicieron posible hacer realidad nuestros sueños en esta Universidad.

De Dailyn:

A mis padres y hermana por su apoyo incondicional, y por haber depositado en mí toda su confianza para poder hoy disfrutar de este gran triunfo.

A todos mis compañeros y amigos que me ayudaron durante todos estos años y me dieron fuerzas para salir adelante, Joaquín, Roger, Belkis, Singh especialmente a mis compañero de tesis Reinier y Dina, que sin nuestro trabajo en equipo no hubiese sido posible obtener este resultado.

A mis familiares por preocuparse por mí, especialmente a mis tíos y primos: Mary, Peña, Omar, Pucho, Lucita, Pepe, Linet, por haberme acogido en su casa durante estos 5 año como una hija más de la casa.

A Danner y a su familia por su amor y confianza.

A todos los profesores que contribuyeron con mi formación y la culminación de esta investigación: Ing. Alain Ramos García, Ing. Kenia Fernández Parra.

De Reinier:

Un trabajo diploma de esta magnitud para que se realice con la calidad requerida, conlleva no solo al trabajo abnegado de ambos autores, sino también al trabajo de un grupo de personas que contribuyeron activamente a la realización del mismo.

A los siguientes compañeros yo les doy mis agradecimientos y las gracias por haberme ayudado en todos estos meses.

Yaniosky Acosta Miranda, Onelio Hierrezuelo (Bolo), Yeikis Vázquez Ruiz, Ing. Alain Ramos García, Jorge Carlos Iglesias y Mairelys Sánchez Boudet.

A todos los que de una forma u otra aportaron su granito de arena en la realización de este trabajo y que en estos momentos olvidamos mencionar.

A Todos, Muchas Gracias.

DEDICATORIA

Para quienes son y serán por siempre nuestra fuente de inspiración
...nuestros Padres y seres queridos.

De Dailyn:

Dedico este trabajo especialmente a mi abuelita Consuelo por estar hoy conmigo disfrutando de este sueño hecho realidad.
A mis padres que son el mayor tesoro que tengo y por haber forjado en mí la persona que soy.
A mi hermana por todo su cariño y por estar siempre ahí cuando la necesito.
A mis amistades, las de antes, las de ahora y las de siempre: Diana, Noelis, Lisi, Serra, Yuyi, Neli, Niurka, Yarelkis, Maribe, Dina, Sara, Lisbel, Ernest, Yuri, Aliozha, Alayn, por haberme soportado durante todos estos años de estudiante.
A mis otros abuelos que aunque no se encuentran conmigo físicamente, siempre los llevo presente y sé que estarían orgullos de mí en este momento.

De Reinier:

Este trabajo diploma se lo dedico a toda mi familia, especialmente a mi madre, abuela, padre y hermana que han sido mi mayor apoyo en esta etapa de mi vida.
A todas las personas que de una forma u otra contribuyeron a mi formación como profesional en estos 5 años de carrera.
A mis amistades o hermanas como yo les llamo: Ilsa, Yadia, Yeilin, Sulema, Karina y Lidicy que me han brindado su apoyo en todo momento.
A un amigo que fue y será siempre para mí como un hermano, que en estos momentos por circunstancias de la vida no se encuentra entre nosotros y que hoy al igual que yo debería estar graduándose. A la memoria de Willy Nicolás Ramos.

RESUMEN

La gestión de la información en los consultorios médicos es manual, por lo que se torna muy lenta de procesar; lo que provoca la pérdida, demora, inconsistencia y desactualización de la información. Para resolver esta problemática, en el presente trabajo se describen los resultados obtenidos con la implementación de una aplicación en ambiente desconectado que agiliza en gran medida el trabajo de los médicos.

En el desarrollo del trabajo se hace referencia al estado del arte, donde se recogen los principales sistemas existentes tanto a nivel nacional como internacional, relacionados con la gestión de información en los consultorios médicos. Se realiza un estudio comparativo sobre las posibles tecnologías y herramientas a utilizar en la implementación del sistema, donde se decidió utilizar como lenguaje de programación Java y como IDE de desarrollo NetBeans. Para lograr una mayor seguridad e integridad de los datos almacenados, se utiliza el gestor de base de datos PostgreSQL e Hibernate para acceder desde Java a los datos.

El sistema informático implementado permite garantizar una mejor calidad, eficiencia y control en la gestión de la información médica de los pacientes, en los consultorios médicos de la Universidad de las Ciencias Informáticas (UCI).

PALABRAS CLAVES

Consultorio Médico, Gestión de Información, Sistema Informático.

ÍNDICE

INTRODUCCIÓN 10

CAPÍTULO I FUNDAMENTACIÓN TEÓRICA 14

1.1 DEFINICIÓN DE LA ATENCIÓN PRIMARIA DE LA SALUD. 14
1.2. LA ATENCIÓN PRIMARIA DE LA SALUD EN CUBA. 14
1.3. SISTEMAS NACIONALES. 17
1.4. ESTILOS ARQUITECTÓNICOS. 18
1.5. ESTÁNDARES DE CODIFICACIÓN. 21
1.6. HERRAMIENTAS Y TECNOLOGÍAS DE DESARROLLO. 22
1.6.1. TÉCNICAS DE PROGRAMACIÓN. 22
1.6.2. LENGUAJE DE PROGRAMACIÓN. 24
1.6.3. ENTORNO DE DESARROLLO INTEGRADO. 27
1.6.4. FRAMEWORKS. 30
1.6.5. GESTOR DE BASE DE DATOS. 31
1.7. PROPUESTA A UTILIZAR. 35

CAPÍTULO II: DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA 37

2.1. VALORACIÓN CRÍTICA DEL DISEÑO PROPUESTO EN EL ANÁLISIS. 37
2.2. CONCEPCIONES GENERALES DE LA ARQUITECTURA. 38
2.3. SEGURIDAD DEL SISTEMA. 41
2.4. ANALISIS DE POSIBLES IMPLEMENTACIONES, COMPONENTES O MÓDULOS YA EXISTENTES QUE PUEDAN SER REUSADOS. 42
2.5. ESTRATEGIAS DE INTEGRACIÓN. 43
2.6. DESCRIPCIÓN DE LOS ALGORITMOS NO TRIVIALES A IMPLEMENTAR. 43
2.7. VISTA DE DESPLIEGUE Y VISTA DE COMPONENTE. 53
2.8. ESTÁNDARES DE CODIFICACIÓN. 55
2.9. ENTIDADES DEL NEGOCIO. 57

CAPÍTULO III VALIDACIÓN DE LA SOLUCIÓN PROPUESTA. 83

3.1. PRUEBAS APLICADAS. 84
3.2. TIPOS DE PRUEBA 85

3.2.1. PRUEBAS DE CAJA BLANCA. 85
3.2.2. PRUEBAS DE CAJA NEGRA. 90
CONCLUSIONES 98
RECOMENDACIONES 99
REFERENCIAS BIBLIOGRÁFICAS 100
BIBLIOGRAFÍA 102
ANEXOS 106
GLOSARIO DE TÉRMINOS 111

INTRODUCCIÓN

El sistema de salud cubano posee, en la Atención Primaria de Salud, una plataforma ideal para articular los avances de las nuevas tecnologías de la información y las comunicaciones en función de hacer más eficiente todo el aparato estratégico y administrativo que lo conforma.

En Cuba, el sistema de salud requiere un manejo racional y eficaz de la información. La salud es una actividad socio-económica de gran importancia, que trabaja directamente con la sensibilidad social al tener responsabilidades directas con la vida de las personas. Es por esto que el sistema de salud, en aras de mejorar su funcionamiento, necesita elaborar software con alto grado de eficiencia en el manejo de datos.

La información que brinden estos sistemas deben ser capaces de respaldar la toma de decisiones, además de ser seguras y precisas. Informatizar el sistema de salud hace explotar al máximo las innovaciones tecnológicas y contribuye a la informatización de la sociedad cubana.

Se considera que la medicina es prácticamente contemporánea con la humanidad. Desde la antigüedad el hombre trató de mantener la salud con prácticas rudimentarias que se fueron aprendiendo y transmitiendo de generación en generación, acumulándose un gran caudal de conocimientos médicos. La acumulación de conocimientos no se podía transmitir ya por las vías orales y generacionales de las tribus, hacía falta que estos conocimientos se almacenaran de alguna manera, para que la información no se perdiera en el paso de una generación a otra, sino que se mantuviera intacta y sin tergiversar.

Así fue como surgió el lenguaje escrito como una de las primeras formas en que se pudo no sólo plasmar la información sino también organizar y darle una cualidad sistemática. El ser humano siguió desarrollándose en muchas otras ciencias y la cantidad de conocimientos acumulados no podía conocerlo un sólo hombre, así que se hacía imprescindible el manejo adecuado de toda la "información". Por lo anteriormente planteado el manejo adecuado de la "información " se convirtió en un elemento vital para el enriquecimiento y desarrollo de todas las ciencias.

Estos procesos de acumulación y asimilación de la información, desde el punto de vista filosófico, han contribuido al desarrollo cultural de la humanidad y sus fuerzas productivas. En general contribuyó a alcanzar estadíos superiores en el desarrollo económico, social y científico de la civilización humana.

Desde los primeros momentos del triunfo de la Revolución, y como uno de los puntos a cumplir del programa del Moncada, el desarrollo de la Salud Pública constituye un elemento que distingue al proceso revolucionario. Innumerables han sido los logros alcanzados por la medicina cubana en estos años de Revolución y los esfuerzos realizados por el Estado Socialista para mantener una atención sanitaria a la altura de países desarrollados.

En Cuba se trabaja intensamente con el objetivo de utilizar las tecnologías de la información y las comunicaciones para apoyar la salud pública. Las acciones que se han emprendido en este sentido parten de reconocer la importancia crucial de la revolución científico-técnica que se vive, pero se han caracterizado sobre todo, por priorizar el factor humano y adecuar estos avances a los problemas reales del país.

Se han creado muchos centros que se encargan de informatizar todas las actividades de la sociedad, en este proceso el área de la salud cobra prioridad por ser la que más beneficios sociales trae. Se mejoran, desde la informática, todo el procesamiento de información en los hospitales y demás áreas de la salud.

La Universidad de las Ciencias Informáticas es una de las instituciones que contribuye a impulsar y crear un mejor camino en la mejora de estos servicios. La misma cuenta con varias facultades y específicamente la facultad 7 se encarga de la informatización del sector de la salud.

A la facultad 7 se le dió la tarea de crear un sistema para la gestión de la información médica de los pacientes en los Consultorios Médicos de la Salud, ya que es un reto para nuestro sistema de salud lograr en años subsiguientes la informatización que permita un buen desarrollo en este campo. El avance vertiginoso de las técnicas de computación hace que surja la necesidad de desarrollar productos informáticos capaces de asimilar, procesar y recuperar la información relacionada con los pacientes y familias del área de atención.

Actualmente no se emplean al máximo las posibilidades que ofrecen las computadoras y las redes de transmisión de datos en nuestros consultorios médicos. Todos los procesos de gestión de información médica (datos del paciente, la creación de historias clínicas, indicaciones de análisis, de medicamentos o tratamientos) se realizan de forma manual, lo que trae como consecuencia la demora, pérdida e inconsistencia de la información, además de hacer mucho más complejo el trabajo del personal de la salud.

El procesamiento manual de la información impide la tenencia de registros actualizados a causa de la demora del llenado de datos y errores de otro tipo que se cometen durante este procedimiento, puesto que no existe ninguna tecnología que gestione dichos procesos. De igual manera, la cantidad de personas atendidas diariamente en los consultorios es reducida para la demanda de atención existente en los mismos.

Dada la situación anterior **el problema** radica en ¿Cómo facilitar la gestión de la información de los pacientes en el consultorio médico de la facultad 7 de la Universidad de las Ciencias Informáticas (UCI)?

El **objeto de estudio** se define como el proceso de gestión de la información médica de los pacientes.

El **campo de acción** enmarca a los procesos de gestión de la información médica de los pacientes que tienen lugar en el consultorio médico de la facultad 7 de la Universidad de las Ciencias Informáticas (UCI).

Para dar solución al problema antes mencionado se propone como **objetivo general**: Implementar un sistema informático que agilice la gestión de la información médica de pacientes en el consultorio médico de la facultad 7 de la Universidad de las Ciencias Informáticas.

Para ello se trazaron las siguientes **tareas de investigación**:

- Valorar los estándares médicos de transmisión de datos a utilizar.
- Determinar los lenguajes y tecnologías de desarrollo.
- Identificar los aspectos teóricos conceptuales referente a la implementación del sistema.
- Realizar el diseño de las clases y la base de datos.

- Valorar los marcos de trabajo (frameworks) que faciliten el acceso y persistencia de los datos.
- Realizar la implementación utilizando los patrones de diseño establecidos en el análisis.

A continuación se da una breve explicación de la estructuración del contenido del presente trabajo de diploma:

El **Capítulo I** titulado "Fundamentación teórica" ofrece los conceptos básicos asociados al dominio del problema. Además brinda el estado del arte en cuanto a los antecedentes históricos del sistema, así como las técnicas, tecnologías, metodologías y software usados en la actualidad o en las que se apoya para la solución del problema que se enfrenta.

El **Capítulo II** denominado "Descripción y análisis de la solución propuesta": se plantea una valoración crítica del diseño propuesto por el analista, un análisis de posibles implementaciones de componentes o módulos ya existentes que puedan ser reusados y las estrategias de integración, una descripción de los algoritmos no triviales a implementar. Análisis de complejidad de los mismos y selección de las estructuras de datos apropiadas para la implementación de estos algoritmos, finalizando con la descripción de las nuevas clases u operaciones necesarias.

En el **Capítulo III** "Validación de la solución propuesta": se muestran las pruebas realizadas para validar la solución. Además se da una explicación detallada de las pruebas de cajas blancas que fueron realizadas al código del software y las pruebas de caja negra que se le realizaron a la interfaz del mismo.

CAPÍTULO I
FUNDAMENTACIÓN TEÓRICA

Este capítulo está dedicado a realizar un análisis detallado del estado del arte de los distintos sistemas de salud tanto nacionales como internacionales, así como las distintas técnicas de programación que existen, se analizarán las tendencias, técnicas, tecnologías, metodologías relacionadas con dichas técnicas y plataformas de desarrollo que la soportan.

1.1. DEFINICIÓN DE LA ATENCIÓN PRIMARIA DE LA SALUD.

La Atención Primaria de Salud (APS) se definió como: «La asistencia esencial, basada en métodos y tecnologías prácticos, científicamente fundados y socialmente aceptables, puesta al alcance de todos los individuos y familias de la comunidad, mediante su plena participación, y a un coste que la comunidad y el país puedan soportar, en todas y cada una de las etapas de su desarrollo, con un espíritu de autorresponsabilidad. [1]

La Atención Primaria es parte integrante tanto del Sistema Nacional de Salud, del que constituye la función central y el núcleo principal, como del desarrollo social y económico global de la comunidad. Representa el primer nivel de contacto de los individuos, la familia y la comunidad con el Sistema Nacional de Salud, llevando lo más cerca posible la atención de salud al lugar donde residen y trabajan las personas y constituye el primer elemento de un proceso permanente de asistencia sanitaria.»[2]

1.2. LA ATENCIÓN PRIMARIA DE LA SALUD EN CUBA.

Desde que se estableció el modelo de atención del médico y la enfermera de la familia, la medicina comunitaria ha obtenido importantes logros. La introducción de dicha modalidad permitió alcanzar, en el país, indicadores de salud muy favorables que contribuyeron a elevar el nivel de salud de la población.

El Sistema de Salud Pública de Cuba está constituido por tres niveles de atención: primaria, secundaria y terciaria. El primer nivel lo constituyen los policlínicos y médicos de familia; el segundo está constituido por los hospitales y el tercero por centros especializados de investigaciones médicas.

El Sistema Nacional de Salud ha desarrollado planes y programas y ha trazado estrategias para elevar la calidad de la atención en salud, tanto del nivel secundario (hospitalario) como primario (policlínico).

La Atención Primaria de Salud (APS) es la piedra angular del Sistema Nacional de Salud de Cuba. Se caracteriza por su cobertura universal, equidad, eficiencia, accesibilidad y la presencia de un escenario docente de excelencia. Dispone de más de treinta mil médicos y enfermeras de la familia, organizados en equipos de trabajo, que junto a otros profesionales constituyen "los guardianes de la salud de nuestra población". [3]

1.3. SISTEMAS INTERNACIONALES.

Después de desarrollar una investigación sobre los diferentes sistemas internacionales existentes en el mundo, se obtuvo como resultados varios sistemas propietarios, con características que cubren las expectativas de los clientes, sin embargo no pueden ser utilizados por el país debido a que es muy elevado el costo de despliegue y soporte de los mismos.

- ✓ **VISUAL MEDIC** es un sistema integral de archivo clínico que le permite controlar de manera sencilla las citas, información clínica y los documentos de sus pacientes. Incluye además opciones para obtener diversos informes estadísticos categorizados, utilizando la información existente en su base de datos. [4]

✓ SOFTWARE DOCTORES-MÉDICOS EN WINDOWS (España).

Su objetivo es la gestión completa de clínicas médicas, seguimiento de historiales de pacientes, creación de recibos, informes, recetas, con formatos modificables, archivo de las fotos o radiografías de sus pacientes, que funcione con el programa en red.

✓ **SISTEMA DE GESTIÓN DE CONSULTORIOS MEDICOS DE GRANDI Y ASOCIADOS. (Datahouse Company) (Argentina).**

Una completa solución de administración de datos y de gestión comercial de su consultorio, integrados de modo transparente. Genera fichas con los datos personales y de contacto de sus pacientes. Lleva un completo registro cronológico de consultas. Deja registradas las historias clínicas de pacientes, e indica los antecedentes de los mismos.

✓ **INTERMEDICAL SOFT vs. 1.0**

Permite el manejo de datos básicos (nombre, dirección, médico, seguro, ocupación, etc.). Permite introducir fotos del paciente, imágenes patológicas relacionadas con el paciente, historia clínica del paciente configurable por especialidades, recetas; incluye manejo de vademécum, estadística de efectividad, medicamentos, próxima fecha y hora de cita. [5]

✓ **MEDICAL CONTROL V2 (México).**

Optimiza el tiempo del médico, asistentes y recepcionistas. MC2 le permite llevar la administración de pagos y pacientes de manera integral. [6]

✓ **MEDICAL CONTROL 1.28**

Es una aplicación para el control, la gestión y la administración de todas las tareas necesarias que se producen en un centro de salud, clínicas o consultorios médicos.

✓ **LANDAMED (Argentina).**

LandaMed, fue desarrollado y pensado para cubrir los requerimientos más actuales de instituciones dedicadas a la salud. Funciona en todas las plataformas Windows adoptando las características que ofrece esta plataforma.

1.3. SISTEMAS NACIONALES.

Después de haber realizado una investigación sobre los distintos sistemas nacionales, se obtuvo como resultado, la existencia de un único software de gestión de información médica en el país conocido como SIDAPS (Sistema Informático para la Dispensarización en la Atención Primaria de la Salud). Dicho producto es propietario, lo que implica costos de desarrollo e instalación muy elevados, por lo que no puede ser utilizado en el país.

✓ **SIDAP.**

Este software es de fácil utilización, posibilita de modo seguro, rápido y eficiente el registro, procesamiento y recuperación de los datos de los pacientes. Para el diseño de la base de datos se utilizó la herramienta *CASE Erwin Ver 2.5*. Se desarrolló la aplicación en versión *stand alone* utilizando como sistema de gestión de bases de datos *Borland Delphi Ver. 7*, compatible con Windows 95/98/NT/XP. [7]

1.4. ESTILOS ARQUITECTÓNICOS.

1.4.1. ARQUITECTURA CLIENTE- SERVIDOR.

La arquitectura cliente/servidor es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos.

Es una arquitectura de procesamientos cooperativos, donde uno de los componentes pide servicios a otro, existiendo una colaboración entre dos o más computadoras conectadas a una red.

IBM define al modelo Cliente/Servidor como: Es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo y/o, a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o clientes, resultan en un trabajo realizado por otros computadores llamados servidores.

Entre las principales características de la arquitectura Cliente/Servidor, se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.
- Centralización del control: los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.
- Escalabilidad: se puede aumentar la capacidad de clientes y servidores por separado.

- Se reduce el tráfico de red considerablemente. Idealmente, el cliente se comunica con el servidor utilizando un protocolo de alto nivel de abstracción.

1.4.2. ARQUITECTURA ORIENTADA A OBJETOS.

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. Estos componentes se basan en principios Orientados a Objetos (OO): encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

Se puede considerar el estilo como perteneciente a una familia arquitectónica más amplia, que algunos autores llaman Arquitecturas de Llamada-y-Retorno (*Call-and-Return*). Desde este punto de vista, los componentes y los objetos, *Call- Return* (C-R) ha sido el tipo dominante en los últimos 20 años.

Como ventajas fundamentales de esta arquitectura se puede decir que permite modificar la implementación de un objeto sin afectar a sus clientes. Así mismo es posible descomponer problemas en colecciones de agentes en interacción. Además un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

1.4.3. ARQUITECTURA EN CAPAS.

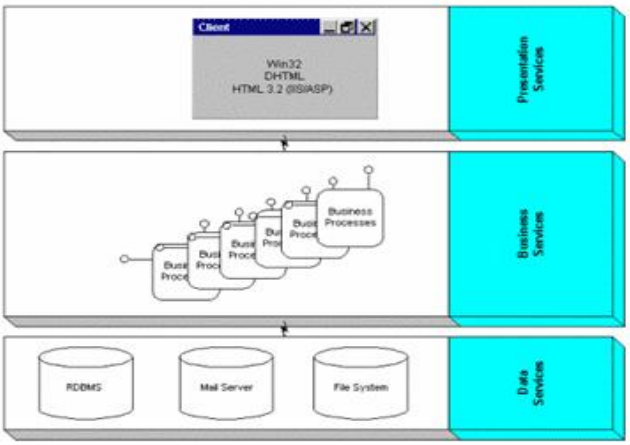
Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia, pues definen el estilo en capas como una organización jerárquica tal, que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Los diagramas de sistemas clásicos en capas dibujaban las capas en adyacencia, sin conectores, flechas ni interfaces; en algunos casos se suele representar la naturaleza jerárquica del sistema en forma de círculos concéntricos.

Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores. Además, ayuda a identificar qué puede reutilizarse, y proporciona una estructura que ayuda a tomar decisiones sobre qué partes comprar y qué partes construir.

Principales estilos de arquitecturas estratificadas de las aplicaciones distribuidas contemporáneas:

- Arquitectura de dos niveles.
- Arquitectura de tres niveles.
- Arquitectura de n niveles.

Las ventajas del estilo en capas son obvias, en primer lugar el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. Además admite muy naturalmente optimizaciones y refinamientos. Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.



La comunidad de software desarrolló la noción de una **arquitectura de tres niveles**. La aplicación se divide en tres capas lógicas distintas, cada una de ellas con un grupo de interfaces perfectamente definido. La primera capa se denomina capa de presentación y normalmente consiste en una interfaz gráfica de usuario de algún tipo. La capa intermedia, o capa de empresa, consiste en la aplicación o lógica de empresa, y la tercera capa, la capa de datos, contiene los datos necesarios para la aplicación.

Fig.1 Arquitectura de tres capas.

La capa intermedia (lógica de aplicación) es básicamente el código al que recurre la capa de presentación para recuperar los datos deseados. La capa de presentación recibe entonces los datos y los formatea para su presentación. Esta separación entre la lógica de aplicación de la interfaz de usuario añade una enorme flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en absoluto la lógica de aplicación siempre que esta presente una interfaz claramente definida a la capa de presentación.

La tercera capa contiene los datos necesarios para la aplicación. Estos datos consisten en cualquier fuente de información, incluido una base de datos de empresa como Oracle o Sybase, un conjunto de documentos XML o incluso un servicio de directorio. Además del tradicional mecanismo de almacenamiento relacional de bases de datos, existen muchas fuentes diferentes de datos de empresa a las que pueden acceder las aplicaciones.

1.5. ESTÁNDARES DE CODIFICACIÓN.

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. [8]

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. [9]

El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un

código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento.

Las técnicas de codificación incorporan muchos aspectos del desarrollo del software. Aunque generalmente no afectan a la funcionalidad de la aplicación, sí contribuyen a una mejor compresión del código fuente. En esta fase se tienen en cuenta todos los tipos de código fuente, incluidos los lenguajes de programación, de marcado o de consulta. [10]

De manera general una técnica de codificación no pretende formar un conjunto inflexible de estándares de codificación. Más bien intenta servir de guía en el desarrollo de un estándar de codificación para un proyecto específico de software.

El estilo de código que se utiliza en Java es la Notación Camello (Mayúsculas y Minúsculas), se usa para denotar variables y parámetros, y en ambos casos la primera palabra debe ser un sustantivo que describa claramente al identificador y las otras palabras.

1.6. HERRAMIENTAS Y TECNOLOGÍAS DE DESARROLLO.

Constituye un objetivo fundamental de los diseñadores de software alcanzar y mantener un nivel técnico acorde con el desarrollo actual en la automatización de la información para la gestión de cualquier proceso a desarrollar, para lo cual es necesario hacer un estudio detallado de las tecnologías a utilizar y las posibilidades de desarrollo que estas brindan, así como los conceptos ligados a estas. A continuación en este epígrafe se describe los principales conceptos, tecnologías y herramientas propuestas para el desarrollo de la solución tratada en el trabajo.

1.6.1. TÉCNICAS DE PROGRAMACIÓN.

Las técnicas de programación tratan de ordenar las actividades de forma que se puedan identificar las relaciones temporales lógicas entre ellas, determinando el calendario o los instantes de tiempo en que debe realizarse cada una.

1.6.1.1. PROGRAMACIÓN NO ESTRUCTURADA.

Este tipo de programación está basada en una secuencia de instrucciones modificando los datos globales en el transcurso de todo el programa. Donde los saltos y el fin de programa no seguían ninguna estructura. Los saltos podían apuntar a cualquier punto del código, lo que ocasionaba que el algoritmo terminara siendo un ovillo indescifrable. Tampoco se podía saber cuándo terminaba.

1.6.1.2. PROGRAMACIÓN PROCEDIMENTAL.

La programación procedimental es un tipo de programación estructurada en donde el código se divide en porciones llamadas "procedimientos" o "funciones".

1.6.1.3. PROGRAMACIÓN MODULAR.

La programación modular está basada en la técnica de diseño descendente, consiste en dividir el problema original en diversos sub-problemas que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema.

1.6.1.4. PROGRAMACIÓN ORIENTADA A OBJETOS.

La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulo más fáciles de escribir, mantener y reutilizar. De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos).

A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

1.6.2. LENGUAJE DE PROGRAMACIÓN.

Son muchos los lenguajes de programación que han aparecido a lo largo de la historia de las computadoras y a pesar de que con cada uno se pueden lograr hacer grandes aplicaciones, como todo, tienen sus ventajas y desventajas.

Una de las desventajas de estos programas es que entre ellos existen maneras muy diferentes de estructurar el código, además de que cada programa maneja sus propias librerías y sintaxis, de igual manera, las funciones en algunos casos se tienen que crear desde cero, lo que conlleva más tiempo a la hora de programar.

1.6.2.1. C#.

Este lenguaje tiene como característica fundamental el hecho de ser muy versátil en su uso y eficiente en su aplicación, conformando un lenguaje que reúne las mejores características de los lenguajes más utilizados.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que **C#** es el **lenguaje nativo de .NET.** [11]

Entre sus características principales se destaca la posibilidad de redefinir operadores e identificar los tipos en tiempo de ejecución (RTTI [*RunTime Type Identification*]); está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Su sintaxis básica se 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.

Aunque forma parte de la plataforma .NET, C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Analizando la superioridad de este lenguaje es importante mencionar que el mismo puede ser implementado lo mismo en un IDE propietario como el Visual Studio .NET o en uno libre (*open source*) como el *SharpDevelop*.

C# es un lenguaje orientado a objetos. 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.

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

En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad. Sin embargo, 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. [12]

C# ha juntado las mejores características de ciertos lenguajes importantes, las ha mejorado, les ha quitado cosas poco útiles y les ha agregado características más versátiles y eficientes en un entorno orientado a la programación de objetos para crear un lenguaje universal.

1.6.2.2. JAVA

Java es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general, ya que es lenguaje libre e independiente de la plataforma. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas de punta.

Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Con

Java se puede programar páginas web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que se desee hacer con acceso a través web se puede hacer utilizando Java.

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el *garbage collector* (reciclador de memoria dinámica).

Además reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- aritmética de punteros
- no existen referencias
- registros (*struct*)
- definición de tipos (*typedef*)
- macros (*#define*)
- necesidad de liberar memoria (*free*)

Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las clases en Java tienen una representación en el *runtime* que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda. [13]

Decir además que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Además, para evitar

modificaciones por parte de los crackers de la red, implementa un método ultra seguro de autentificación por clave pública.

El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto ningún objeto se crea y almacena en memoria, sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento en que se interpreta, con el nivel de detalle y de privilegio que sea necesario.

Al ser multihilo, Java permite muchas actividades simultáneas en un programa. El beneficio consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. De igual manera las imágenes se pueden ir trayendo en un hilo de ejecución independiente, permitiendo que el usuario pueda acceder a la información de la página sin tener que esperar por el navegador.

1.6.3. ENTORNO DE DESARROLLO INTEGRADO.

Un entorno de desarrollo integrado o *Integrated Development Environment* (IDE), es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

1.6.3.1. ECLIPSE.

Eclipse es un Entorno de Desarrollo Integrado (IDE) que en un principio fue diseñado y desarrollado por IBM y que luego fue lanzada a la comunidad de software libre, y que se distribuye mediante una licencia de código abierto, la Eclipse Public License (EPL).

La plataforma de Eclipse integra herramientas de desarrollo, con una arquitectura abierta y basada en plug-ins. Además, da soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado.

La arquitectura de plug-ins permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones accesorias. Conservan el registro de las versiones, generan y mantienen la documentación de cada etapa del proyecto.

Como principales características Eclipse posee un editor visual con sintaxis coloreada, además de modificar e inspeccionar valores de variables. También avisa de los errores cometidos mediante una ventana secundaria, y depura el código que resida en una máquina remota. Eclipse permite agrupar código escrito y mostrado visualmente como una estructura empaquetada para que sea fácil poder seguir un código escrito. [14]

Además este IDE es soportado por los principales sistemas operativos:

- Linux
- Windows
- Solaris 8 (SPARC/GTK 2)
- Mac OSX –Mac/Carbon

En los últimos tiempos, la popularidad de Eclipse ha subido enormemente entre los desarrolladores no sólo de Java, sino también de los otros lenguajes a los que Eclipse da soporte, pues es una plataforma de herramientas universal y portable que proporciona un marco de trabajo o framework para desarrollar aplicaciones y herramientas.

1.6.3.2. NETBEANS.

NetBeans es un Entorno de Desarrollo gratuito, y de código abierto para desarrolladores de software. En esta versión se tiene al alcance todas las herramientas necesarias para crear aplicaciones profesionales para entornos de escritorio, empresa, web, ya sea en C/C++, Java e incluso Ruby. El IDE ha sido desarrollado para distintas plataformas como Linux, MacOS X, Solaris y también Windows.

Este entorno permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos, debido a que los módulos pueden ser desarrollados independientemente.

Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis y por si fuera poco sus funcionalidades son ampliables mediante la instalación de packs. [15]

Ventajas:

1. Incorpora la línea de tecnología de Sun dentro del entorno.
2. No hay que buscar plug-ins por todas partes, por lo regular todo se encuentra en la misma caja.
3. La estructura de los proyectos está basada en ant, por lo tanto es muy personalizable, por si después de empaquetar un proyecto se quiere enviar en ftp, se puede hacer fácilmente con ant.
4. Developer Collaboration: Es un plug-in que como su nombre lo indica, permite hacer desarrollos en equipo. Se puede modificar el mismo archivo a la vez y ver los cambios en tiempo real, se puede estar chateando o enviar pedazos de códigos.

La plataforma de NetBeans ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación.

Entre las **características** de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas)
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Framework basado en asistentes (diálogo paso a paso).

El NetBeans IDE es un entorno de desarrollo - una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender este IDE, que es un producto libre y gratuito sin restricciones de uso. [16]

1.6.4. FRAMEWORKS.

Un *framework* es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Además representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

1.6.4.1. SWING.

Con el *NetBeans Swing GUI builder*, tanto los usuarios finales como los desarrolladores obtienen facilidad de uso adicional. Swing simplifica drásticamente la creación de interfaces gráficos de usuario para aplicaciones cliente y de Internet avanzadas, permite a los programadores manejar diferentes guías de estilo en varias plataformas, y asegura la adaptación de las aplicaciones a una gran diversidad de idiomas.

Swing Application es un *framework* excelente, con un buen diseño que facilita tareas normalmente tediosas al crear una aplicación *desktop* en *Swing*. Además una gran ventaja de este sobre otros *frameworks* es por ejemplo los *open swings*, el *Swing Application Framework* es mucho menos intrusivo y se puede integrar fácilmente con *Beans Binding*, aporta las partes más básicas de una aplicación de escritorio de una manera sencilla, que permite empezar de manera rápida una aplicación. [17]

1.6.4.2. HIBERNATE.

Hibernate es una capa de persistencia objeto/relacional y un generador de sentencias SQL. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se puede generar la base de datos en cualquiera de los entornos soportados: Oracle, DB2, MySQL, etc... Y lo más importante de todo, es *open source*, lo que supone, entre otras cosas, que no se tiene que pagar nada por adquirirlo. [21]

Uno de los posibles procesos de desarrollo consiste en, una vez que se tiene el diseño de datos realizado, mapear este a ficheros XML siguiendo la Definición de Tipo de Documento (Document Type Definition (DTD)) de mapeo de Hibernate. Desde estos se puede generar el código de nuestros objetos persistentes

en clases Java y también crear la base de datos independientemente del entorno escogido. Se integra en cualquier tipo de aplicación justo por encima del contenedor de datos.

Como todas las herramientas de su tipo, busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos. Permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información, le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la Programación Orientada Objetos. Además convertirá los datos entre los tipos utilizados por Java y los definidos por SQL.

Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución. Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Ofrece también un lenguaje de consulta de datos llamado HQL (*Hibérnate Query Language*), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria").

Para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente *Hibernate Annotations* que implementa el estándar JPA, que es parte de esta plataforma.

1.6.5. GESTOR DE BASE DE DATOS.

Un sistema gestor de base de datos es una aplicación capaz de manejar un conjunto de datos de manera eficiente y cómoda, asegurando su integridad, confidencialidad y seguridad.

1.6.5.1. MySQL.

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones y con la característica que es un *software* libre.

Este gestor de bases de datos es, probablemente el más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida, en parte, a que existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.[18]

Las principales **características** de este gestor de bases de datos son las siguientes:

Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo. Además soporta gran cantidad de tipos de datos para las columnas. Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc). Posee gran portabilidad entre sistemas, y puede trabajar en distintas plataformas y sistemas operativos.

Cada base de datos cuenta con 3 archivos: Uno de estructura, uno de datos y uno de índice y soporta hasta 32 índices por tabla. MySQL tiene un flexible sistema de contraseñas (passwords) y gestión de usuarios, con un muy buen nivel de seguridad en los datos, y su principal objetivo es velocidad y robustez. [19]

Principales **ventajas**:

- Velocidad al realizar las operaciones, lo que le hace uno de los gestores con mejor rendimiento.
- Bajo costo en requerimientos para la elaboración de bases de datos, ya que debido a su bajo consumo puede ser ejecutado en una máquina con escasos recursos sin ningún problema.
- Facilidad de configuración e instalación.
- Soporta gran variedad de Sistemas Operativos.
- Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.
- Conectividad y seguridad.

Principales **desventajas**:

- Un gran porcentaje de las utilidades de MySQL no están documentadas.
- No es intuitivo, como otros programas (ACCESS).

MySQL es sin dudas uno de los gestores de base de datos libres más empleados.

1.6.5.2. PostgreSQL.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional. Fue el pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, incluido, más tarde en otros sistemas de gestión comerciales. Es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos. [20]

Las principales **características** de este gestor de bases de datos son las siguientes:

PostgreSQL es un servidor libre que soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, etc. Además incorpora una estructura de datos, los arreglos. Permite la declaración de funciones propias, así como la definición de disparadores (triggers). De igual manera incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales. También permite la gestión de diferentes usuarios, como los permisos asignados a cada uno de ellos.

Principales **ventajas**:

- Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (*constraints*) y disparadores (*triggers*).

- Se han añadido funcionalidades en línea con el estándar SQL92, incluyendo claves primarias, identificadores entrecomillados, forzado de tipos cadena literal, conversión de tipos y entrada de enteros binarios y hexadecimales.
- Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora de rango amplio y soporte para tipos geométricos adicionales.
- La velocidad del código del motor de datos ha sido incrementada aproximadamente en 40%, y su tiempo de arranque ha bajado el 80%.
- Establece condiciones o *CHECKs* para validar las entradas de datos.
- Permite transacciones, es decir, múltiples operaciones de tabla o registros de manera segura.
- Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos, equiparándolo con los gestores de bases de datos de alto nivel, como puede ser Oracle.

Los lenguajes que utiliza PostgreSQL son los siguientes:

- Un lenguaje propio llamado PL/PgSQL (similar al PL/SQL de Oracle), C, C++, Gambas.
- Java PL/Java web, PL/Perl, plPHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl, PL/Scheme.
- Lenguaje para aplicaciones estadísticas R through PL/R.

Principales **desventajas**:

- La velocidad de respuesta que ofrece este gestor, con bases de datos relativamente pequeñas es un poco deficiente.
- Consume gran cantidad de recursos.
- Tiene un límite de 8K por fila, aunque se puede aumentar a 32K, con una disminución considerable del rendimiento.

PostgreSQL es sin dudas uno de los dos gestores de base de datos libres más empleados.

1.7. PROPUESTA A UTILIZAR.

Después de haber realizado un amplio estudio sobre las diferentes herramientas a utilizar, se propone para el desarrollo de esta aplicación utilizar como lenguaje de programación JAVA, en primer lugar por ser libre, y además porque proporciona numerosas ventajas que favorecen en gran medida la elaboración del producto. Teniendo en cuenta que se va a desarrollar una aplicación de escritorio, se decidió utilizar la plataforma NetBeans, quien trae incluido muchos plugins configurados, además es un Entorno de Desarrollo Integrado (IDE) muy simple que proporciona numerosas ventajas.

También se realizó un estudio con los diferentes gestores de base de dato, donde se pudo determinar que ambas herramientas poseen características muy satisfactorias para la elaboración de este software. Sin embargo, debido a las peticiones realizadas por los clientes, se decidió escoger como gestor de Base de Datos de dicha aplicación a PostgreSQL teniendo en cuenta las ventajas y la seguridad de almacenamiento de información que este proporciona.

Teniendo en cuenta que Swing es un nuevo paquete gráfico que ha aparecido en la versión 1.2 de Java, se determinó utilizarlo para el desarrollo de este software, ya que está compuesto por un amplio conjunto de componentes de interfaces de usuario que funcionan en el mayor número posible de plataformas. Además se utilizó como puente entre la aplicación y la base de datos el framework Hibernate, puesto que sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.

CONCLUSIONES.

En este capítulo se describió de manera general el estado del arte, donde se analizaron los softwares que existen a nivel nacional e internacional, de los cuales se determinó que no es factible su utilización debido a que todos contienen licencias propietarias, por lo que cumple con la política de emigración a software libre que se está ejerciendo en el país. También se han caracterizado las herramientas, tecnologías y lenguajes a emplear para el diseño e implementación de un sistema que posibilite la gestión de la información de los pacientes en los consultorios médicos.

CAPÍTULO II:
DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA

En este capítulo se aborda el análisis del diseño propuesto por los analistas. Se analizan las posibles implementaciones de módulos y componentes, así como la reutilización de los existentes. Se harán descripciones de las clases, los tipos de datos y las operaciones que se implementen para dar solución al problema.

2.1. VALORACIÓN CRÍTICA DEL DISEÑO PROPUESTO EN EL ANÁLISIS.

Del diseño propuesto por los analistas se pudo extraer las clases fundamentales que deben ser definidas para que el sistema funcione satisfactoriamente, así como los atributos y métodos que deben tener las mismas, creando una entrada apropiada y un punto de partida para las actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.

Unido a esto se encuentran los diagramas de interacción, que explican la colaboración que existe entre las clases y cómo son llamados los métodos y sentencias dentro de cada una, de los cuales se obtuvo la información necesaria para conocer el orden de las acciones a implementar.

Se obtuvo además, diagramas de clases de diseño que mostraron la parte estática del sistema, la representación de las clases y sus relaciones, cosa que facilita en gran medida la implementación de las mismas dada sus definiciones.

El diseño propuesto fue creado siguiendo patrones, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, permitiendo llevar a cabo la implementación clara y limpia del módulo bajo patrones como los Patrones de Asignación de Responsabilidades (GRASP).

Estos patrones se les aplican a las diferentes clases que se definen en el diseño. Dentro de este grupo se identifican cinco patrones muy utilizados: Experto, Creador, Alta Cohesión, Bajo Acoplamiento y el Controlador. Estos patrones se les aplican a las clases definidas en el diseño, distribuyendo responsabilidades entre las mismas de forma tal que no existan muchas relaciones, que no se sobrecargue de métodos a una clase en específico pudiendo acomodarlos en otras, entre otras mejoras que brinda el uso de este grupo de patrones.

2.2. CONCEPCIONES GENERALES DE LA ARQUITECTURA.

El sistema se desarrollará utilizando RUP como metodología. Esta metodología define tres puntos fundamentales que guían el diseño de la arquitectura, estos son:

- Guiado por los casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

Estos puntos garantizan que la solución satisfaga las necesidades del cliente, ya que al centrar el desarrollo en los casos de uso permite concebir el sistema en módulos teniendo en cuenta para cada uno de ellos el conjunto de operaciones que dada su prioridad y complejidad son escogidas para el desarrollo de la primera iteración del sistema. Es válido que con la primera iteración no culmina el desarrollo, en las iteraciones restantes y definidas en el plan de desarrollo del proyecto, se definen y estructuran las restantes necesidades del sistema y se refinan las desarrolladas en iteraciones anteriores.

2.2.1. ORGANIGRAMA DE LA ARQUITECTURA.

El sistema desarrollado para los Consultorios Médicos entra dentro de la categoría de Software de Gestión de Información Hospitalaria. Este debe poseer una base de datos que le permita el almacenamiento de la información, que en este tipo de sistema debe poseer el grado máximo de confiabilidad, seguridad y garantía, ya que los principales beneficiados o afectados con este sistema es el propio hombre.

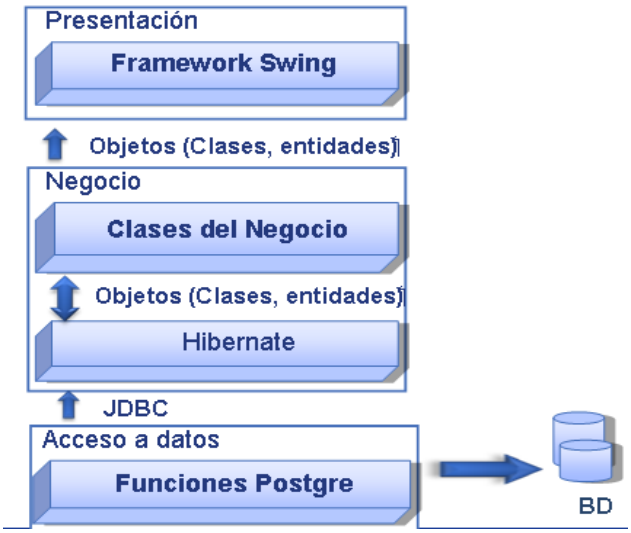


Fig. 2 Estructura Modular del Sistema.

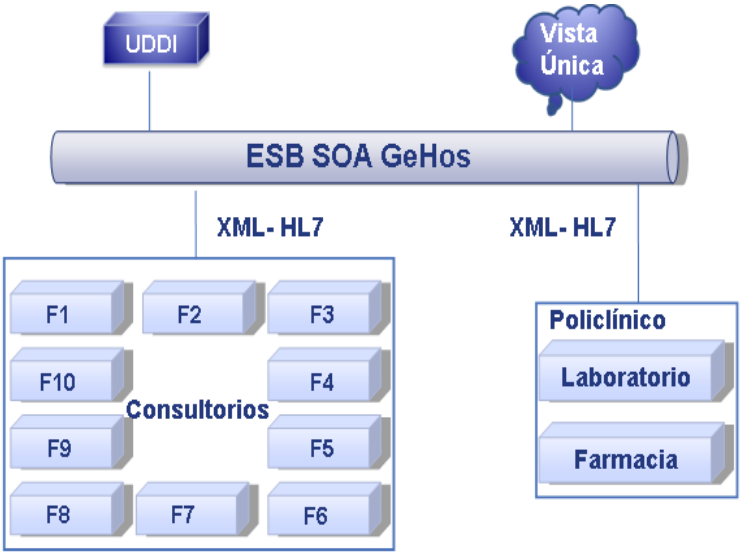


Fig. 3 Estructura Global del Sistema.

2.2.2. COMPONENTES O ELEMENTOS.

Los principales componentes que distinguen la estructura modular son:

- Presentación:

Esta capa contiene las interfaces necesarias para que el usuario y el sistema intercambien toda la información necesaria para el proceso de gestión. Interactúa con la capa inferior, mediante invocación de los métodos que conformar la lógica del negocio, produciendo un intercambio de objetos, utilizando el framework Swing.

- Negocio:

Esta capa se divide en dos subcapas encargadas de resolver la lógica del negocio.

Clases del Negocio: Almacena todas las clases encargadas de representar la lógica del negocio, y se controla la seguridad en cada invocación de los métodos. También interactúa con la subcapa Hibernate quien trabaja con la librería Top link para acceder a los datos. De igual manera que el anterior se produce el intercambio de objetos.

Hibernate: Es una subcapa con la función de acceder al los datos y es la encargado de ejecutar la lógica de acceso a datos.

- Acceso a Datos:

Se encuentra ubicada en el servidor de Bases de datos, donde se utiliza el gestor PostgreSQL conjuntamente con Hibernate para lograr el acceso y solicitudes sobre el servidor, además de englobar toda la lógica de acceso a datos.

Principales componentes que se distinguen en la estructura global:

- ESB SOA GeHos:

Conjunto de Servicios WEB que cubren las funcionalidades exigidas por la lógica de negocio; utilizados para facilitar la integración con sistemas externos ó módulos del sistema, que no se puedan conectar directamente con el negocio de los restantes módulos por problemas de configuración de la red o localizaciones de estos. La información facilitada por estos consiste en mensajes HL7 sobre XML construidos y revisados.

- Consultorios Médicos:

El sistema interactúa con todos los consultorios existentes en la universidad.

- Policlínico:

El sistema también interactúa con el policlínico, específicamente con los módulos de Farmacia y Laboratorio, quienes actualmente se encuentran en construcción. De igual manera son generados una

serie de reportes por cada uno de los registros que contienen toda la información de los pacientes, y estos son exportados mediante el correo electrónico.

- ❖ UDDI:

El intercambio de información se realiza a través de servicios web, utilizando esta herramienta con una conexión XML- HL7. Decir además que la implementación del ESB Gehos no se encuentra dentro de nuestro campo de acción, solo se muestra esta vista global con el objetivo de que se comprenda el alcance del sistema en caso de que se implementen las capas de servicios que garanticen el intercambio de información de los consultorios con el policlínico y el repositorio central de datos, a los cuales este sistema surte de datos e información sobre los pacientes.

2.3. SEGURIDAD DEL SISTEMA.

En la actualidad, la importancia de la seguridad en aplicaciones es crucial. El sistema en desarrollo cuenta con tres usuarios: el administrador del sistema, el médico y la enfermera, cada uno con los privilegios necesarios para acceder a sus páginas.

- ➢ El administrador: Puede acceder a todas las formas que tiene el sistema, ya que cuenta con los permisos necesarios para acceder al mismo.

- ➢ El médico: Puede acceder únicamente a las interfaces que tienen que ver con él es decir:

- Consulta.

- Remisión a Especialista.

- Certificado Medico.

- Certificado por Tarjeta.

- Complementarios.

- Receta Médica y Método.

- Historia Clínica.

➢ La enfermera: Puede acceder solamente a la parte de enfermería.

- Pedidos de Farmacia y Miscelánea.
- Proceder de Enfermería.
- Historia Clínica.

2.4. ANALISIS DE POSIBLES IMPLEMENTACIONES, COMPONENTES O MÓDULOS YA EXISTENTES QUE PUEDAN SER REUSADOS.

El Sistema de Gestión Hospitalaria está compuesto por varios módulos, de los cuales el consultorio médico necesita intercambiar información.
Estos módulos son:

Laboratorio Clínico: Proveedor de la información referente a los resultados de los análisis que se le han indicado al paciente.

Farmacia: Abastecedor y controlador de los medicamentos que están disponibles, así como los materiales gastables implicados en cada una de las consultas médicas.

Policlínico: Controla todas las historias clínicas de cada uno de los consultorios de la universidad, a demás es a este módulo al que se le envían diariamente los reportes de cada una de las consultas realizadas, y dentro de él se encuentran los módulos de *Farmacia* y *Laboratorio* que tienen una estrecha relación con el consultorio.

También el sistema necesita estar integrado con algunos módulos pertenecientes a la Atención Primaria de la Salud (APS) como es:

RAD: El Registro de Actividades Diarias, como su nombre lo indica, en él se registra diariamente cada una de las actividades que realiza el médico en el consultorio.

2.5. ESTRATEGIAS DE INTEGRACIÓN.

Todo el código dentro un mismo componente se comunica mediante llamadas a métodos o eventos de forma directa. La información que es transmitida debe cumplir con los estándares internacionales que hay establecidos para facilitar la integración entre nuevos componentes y otros sistemas hospitalarios. La base de datos solamente es accedida directamente por las clases del negocio, pues el resto de los componentes o sistemas externos solamente se comunican con el sistema y con los datos que este maneja a través de la capa de negocio, nunca va a ser directamente con la base de datos, además de cumplir con ciertos estándares y normas de seguridad.

2.6. DESCRIPCIÓN DE LOS ALGORITMOS NO TRIVIALES A IMPLEMENTAR.

Los Pedidos de Miscelánea y Farmacia son uno de los servicios más importantes que son automatizados por el sistema Consultorio Médico.

El Pedido de Miscelánea es un tipo de almacén donde cada consultorio obtiene materiales importantes para el buen desenvolvimiento de la unidad, en dicho local se almacenan materiales tales como: gasas, algodón, torundas, etc.

En este servicio se recogen datos como la facultad que solicita el pedido, el día y la cantidad que se solicita. Después que el almacén manda el pedido al consultorio, se recoge el despacho ya que la cantidad que dé, no sea la misma que la que haya físicamente en el almacén.

Para el Pedido de Farmacia se realiza el mismo procedimiento lo que con medicamentos. Cada consultorio hace los pedidos de los medicamentos a la farmacia del policlínico, de donde se obtienen los medicamentos solicitados.

A continuación se muestra el código correspondiente a *Adicionar Pedido de Miscelánea*:

```
public void AdicionarPedidoMiscelaneo(TbnUnidadSalud unidad, Integer cantidad, String tipo, Date dia,
TbnPresentacion presentacion,
   TbnProductos productos, Integer despacho, Double unidadmedida)
   {
```

```
        Pedido ped = new Pedido();
        ped.setId(identificadorPedido());
        ped.setUnidadSaludid(unidad);
        ped.setCantidadOrden(cantidad);
        ped.setTipoOrden(tipo);
        ped.setDiaPedido(dia);
        ped.setTbnPresentacionid(presentacion);
        ped.setTbnProductosid(productos);
        ped.setDespacho(despacho);
        ped.setUnidadMedida(unidadmedida);
        salvar.persistencia(ped);
            }
```

- Método Visual de *Adicionar Pedido de Miscelánea:*

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
 {
       Validacion v = new Validacion();
       save s = new save();

   Adicionar Pedido de Miscelánea:
       if ((tipo.matches("Miscelanea"))) {

          if ( v.validaCampoSelect(consultorio.getSelectedItem().toString(), true, "")
                && v.validaCampoNumeros(cantidad.getText(), true, "", 3, false)
                && v.validaCampoSelect(presentacion.getSelectedItem().toString(), true, "")
                && v.validaCampoSelect(producto.getSelectedItem().toString(), true, "")
                && (diaPedido.getDate() != null)) {
```

```
        if(!((consultorio.getSelectedItem() == "--Seleccione--") && (presentacion.getSelectedItem() == "--Seleccione--") && (producto.getSelectedItem() == "--Seleccione--")))
        {

        TbnUnidadSalud unidad =  (TbnUnidadSalud) entityManager1.createQuery("Select p From TbnUnidadSalud p where p.nombre =:UnidadSalud ").
                setParameter("UnidadSalud",consultorio.getSelectedItem()).getSingleResult();
        TbnPresentacion  presenta = (TbnPresentacion) entityManager1.createQuery("Select p From TbnPresentacion p where p.nombre =:presentacion ").
                setParameter("presentacion", presentacion.getSelectedItem()).getSingleResult();
        TbnProductos  product = (TbnProductos) entityManager1.createQuery("Select p From TbnProductos p where p.nombre =:producto ").
                setParameter("producto", producto.getSelectedItem()).getSingleResult();

        c.AdicionarPedidoMiscelaneo(unidad, Integer.parseInt(cantidad.getText()), tipo, diaPedido.getDate(), presenta, product, 0, 0.0);
         LimpiarPedido();
          JOptionPane.showMessageDialog(null, "Se ha adicionado satisfactoriamente");
        }
        else
        {
            JOptionPane.showMessageDialog(null, "Faltan datos obligatoros por entrar por entrar");
        }
      } else {
        if( v.validaCampoSelect(consultorio.getSelectedItem().toString(), true, "") == false
                || v.validaCampoNumeros(cantidad.getText(), true, "", 3, false) == false
                || v.validaCampoSelect(presentacion.getSelectedItem().toString(), true, "") == false
                || v.validaCampoSelect(producto.getSelectedItem().toString(), true, "") == false
                || (diaPedido.getDate() != null) == false) {
```

```
            JOptionPane.showMessageDialog(null, "Faltan datos obligatorios por entrar");
          }

        }
      }
      else {
```

- *Adicionar Pedido de Farmacia*:

```
        if ( v.validaCampoSelect(consultorio.getSelectedItem().toString(), true, "")
            && v.validaCampoNumeros(cantidad.getText(), true, "", 3, false)
            && v.validaCampoSelect(presentacion.getSelectedItem().toString(), true, "")
            && (diaPedido.getDate() != null)) {

            if(!((consultorio.getSelectedItem() == "--Seleccione--") && (presentacion.getSelectedItem() ==
"--Seleccione--") && (medicamentos.getSelectedItem() == "")))
          {

          TbnUnidadSalud unidad =  (TbnUnidadSalud) entityManager1.createQuery("Select p From
TbnUnidadSalud p where p.nombre                =:UnidadSalud ").
              setParameter("UnidadSalud",consultorio.getSelectedItem()).getSingleResult();
        TbnMedicamento  medic = (TbnMedicamento) entityManager1.createQuery("Select p From
TbnMedicamento p where p.descripcion =:medicamentos ").
            setParameter("medicamentos", medicamentos.getSelectedItem()).getSingleResult();
        TbnPresentacion  presenta = (TbnPresentacion) entityManager1.createQuery("Select p From
TbnPresentacion p where p.nombre =:presentacion ").
            setParameter("presentacion", presentacion.getSelectedItem()).getSingleResult();

        c.AdicionarPedidoFarmacia(unidad, Integer.parseInt(cantidad.getText()), tipo, diaPedido.getDate(),
presenta, medic, 0, 0.0);
```

```
            LimpiarPedido();
            JOptionPane.showMessageDialog(null, "Se ha adicionado satisfactoriamente");
        }
            else
            {
                JOptionPane.showMessageDialog(null, "Faltan datos obligatoros por entrar por entrar");
            }
        }
        else {
            if ( v.validaCampoSelect(consultorio.getSelectedItem().toString(), true, "") == false
                || v.validaCampoNumeros(cantidad.getText(), true, "", 3, false) == false
                || v.validaCampoSelect(presentacion.getSelectedItem().toString(), true, "") == false
                || (diaPedido.getDate() != null) == false) {
            JOptionPane.showMessageDialog(null, "Faltan datos obligatoros por entrar por entrar");
            }
        }
    }

}
```

2.6.1. ANÁLISIS DE COMPLEJIDAD.

La Complejidad Ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. La métrica, propuesta por Thomas McCabe en 1976, se basa en la representación gráfica del flujo de control del programa. De dicho análisis se desprende una medida cuantitativa de la dificultad de prueba y una indicación de la fiabilidad final. [22]

Cuando se utiliza en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa

y proporcionando el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Es una de las métricas de software más ampliamente aceptada, ya que ha sido concebida para ser independiente del lenguaje. Se ha medido un gran número de programas, de modo de establecer rangos de complejidad que ayuden al ingeniero de software a determinar la estabilidad y riesgo inherente de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad. [23]

Se suele comparar la complejidad ciclomática obtenida contra un conjunto de valores límite como se observa en la **tabla 1**.

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa Simple, sin mucho riesgo.
11-20	Más complejo, riesgo moderado.
21-50	Complejo, Programa de alto riesgo.
+50	Programa no testeable, Muy alto riesgo.

Tabla 1: **Complejidad ciclomática vs evaluación de riesgo**.

Para conocer la complejidad del algoritmo es necesario calcular la complejidad ciclomática del mismo, para hacer dicho cálculo es necesario primero tener el código o el diseño del algoritmo, luego enmarcar cada instrucción del código con un número, que representa cada lugar del camino que puede seguir la secuencia del algoritmo. A continuación se representa el código con sus instrucciones enmarcadas:

```
public List<Historial Enfermedades> Buscar () //es el método encargado de buscar por parámetros.
```

```
{
    if (! Parentesco.getSelectedItem ().toString ().isEmpty ()) (1)
   {
     EntityManagerFactory pac = Persistence.createEntityManagerFactory ("tesisPU");    (1)
          EntityManager em = pac.createEntityManager ();     (1)
        Listpa = em.createQuery ("SELECT p FROM HistorialEnfermedades p where
     (p.historiaClinicaid.pacientehc.hc=: hc) AND (p.tbnParentescoid.nombre =:
     Parentesco)").setParameter("hc",Identif).setParameter("Parentesco",Parentesco.getSelectedItem()).ge
     tResultList();          (1)
    if (listpa.size() > 0)      (2)
      {
        s = new String[listpa.size ()] [7];     (2)
       int i = 0;     (2)
    for (HistorialEnfermedades p: listpa)     (3)
      {
          String[] str = {p.getTbnEnfermedadesid ().getNombre()};     (3)
          s[i] = str;       (3)
          i++;      (4)
       }
```

```
            }     (5)
        else
    {
JOptionPane.showMessageDialog (null, "No tiene ninguna enfermedad");     (6)
     }
          }     (7)
      else
     {
JOptionPane.showMessageDialog (null, "Debe escoger un parentesco");     (8)
     }
       return listpa;   (9)
     }
```

Después de este paso, es necesario representar el grafo de flujo asociado (*Figura 4*), en el cual se representan distintos componentes como son los círculos que se denominan NODOS y representan una o varias sentencias procedimentales. Las flechas se denominan ARISTAS y representan flujo de control. Una arista debe terminar en un nodo, aún cuando éste no represente ninguna sentencia procedimental. Las áreas delimitadas por aristas y nodos se denominan regiones.

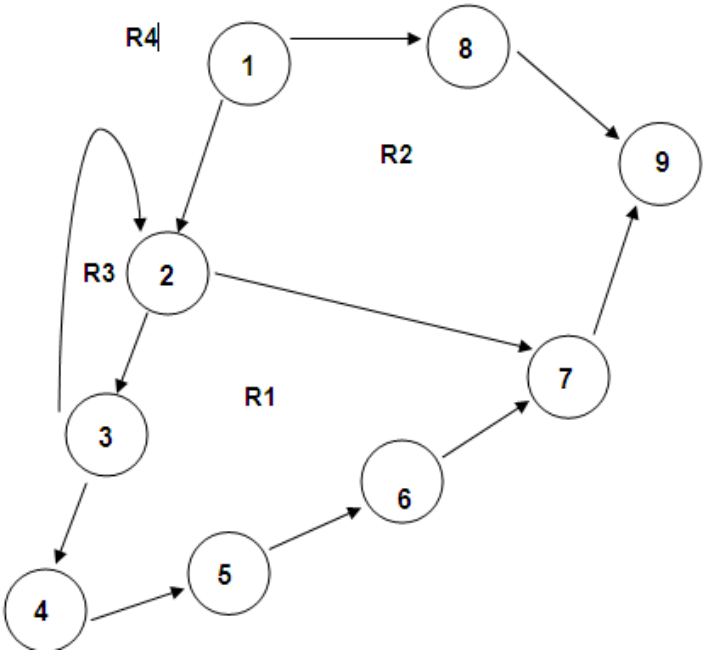


Fig. 4. Grafo del flujo del algoritmo.

Seguidamente a la construcción del grafo de flujo se procede a efectuar el cálculo de la complejidad ciclomática del código, el cálculo es necesario efectuarlo mediante tres vías, para concluir que fueron correctos es necesario que el resultado sea el mismo, las fórmulas para calcular son las siguientes:

V (G) = A – N + 2

Donde A es el número de aristas en el grafo, N es el número de nodos. V se refiere al número ciclomático en teoría de grafos y G indica que la complejidad es una función del grafo.

V (G) = 11 – 9 + 2 = 4.

V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).
V (G) = 3 + 1= 4.

V (G) = R
Siendo “R” la cantidad total de regiones, para cada fórmula “V (G)” representa el valor del cálculo.
V (G) = 4.

Después de haber realizado el cálculo por las tres vías antes expuestas, se puede concluir que el algoritmo representado anteriormente tiene una complejidad ciclomática de 4, dando una visión de que existen a lo sumo 4 caminos lógicos por donde recorrer el algoritmo. Al tener una complejidad de 4, la evaluación de riesgo dice que es un programa simple, sin mucho riesgo.

2.6.2. SELECCIÓN DE LAS ESTRUCTURAS DE DATOS APROPIADAS PARA LA IMPLEMENTACIÓN DE ESTOS ALGORITMOS. DESCRIPCIÓN DE LAS CLASES QUE SE UTILICEN PARA REPRESENTAR COMPUTACIONALMENTE DICHA ESTRUCTURA.

Para llevar a cabo la implementación de la funcionalidad de Pedidos tanto de Farmacia como de Miscelánea, fue necesario crear las Clases de *Entidades-Pedidos* la cual contiene todo la conexión para la entrada rápida y fácil de datos, así como todos los atributos fundamentales para su correcta funcionalidad.

También es creada la clase llamada *Save* que es la encargada de la persistencia de los datos de la clase anterior.

Para listar los resultados se utilizó la lista genérica de JAVA.

La lista genérica es una de las nuevas clases de JAVA, que está dentro del código fuente del lenguaje en cuestión. Como su propio nombre indica, permite listar cualquier dato, desde un simple listado de String hasta un listado de la clase más compleja que se haya creado. Permite funciones muy útiles para ordenar, buscar un índice, comparar, etc.

La lista es una especie de arreglo que se va redimensionando conforme a las necesidades. Al crear la variable *List<T>* se inicializa su capacidad, que aumenta conforme la lista va creciendo, pero esto es totalmente transparente. El implementador puede hacer la lista tan grande como necesite, y listarlo fácilmente.

La lista permite dado el valor de una posición obtener el elemento que se encuentra en la misma sin tener que recorrerla innecesariamente, admite insertar un objeto en la posición deseada desplazando los demás a la posición inmediata superior, también permite eliminar, adicionar y muestra mediante la propiedad *"Count"* la cantidad de elementos que contiene.

2.7. VISTA DE DESPLIEGUE Y VISTA DE COMPONENTE.

2.7.1 VISTA DE DESPLIEGUE.

Un diagrama de despliegue muestra la distribución física del sistema (ordenadores, dispositivos).

La vista de despliegue muestra la configuración de la plataforma en tiempo de ejecución para un ambiente de desarrollo. Los nodos utilizados son:

- Una PC- Cliente: que es donde va a estar instalado el software.
- Servidor de Base de Datos: es donde se encuentra almacenada la base de dato con toda la información referente a los pacientes.
- Impresora: con este dispositivo se va a imprimir en caso de que sea necesario, ya sea una receta médica, hoja de cargo o cualquier otro documento que lo requiera.

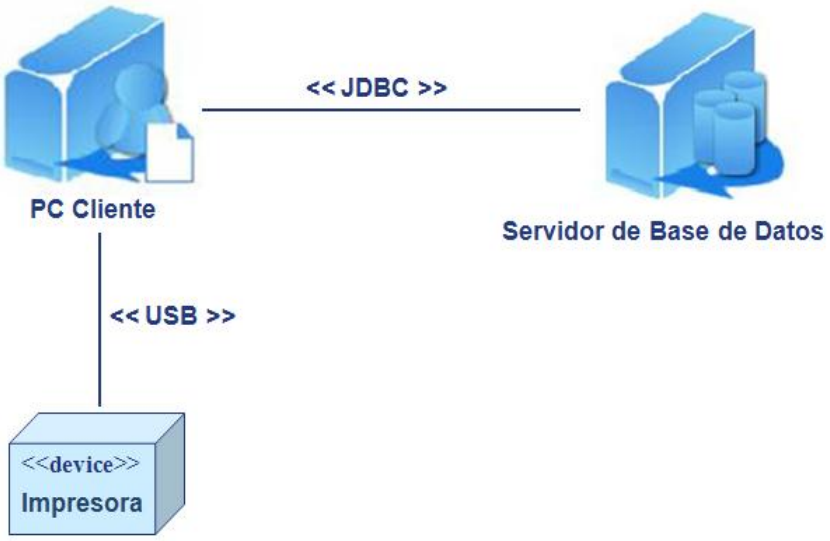


Fig. 5 Vista de despliegue.

2.7.2. VISTA DE COMPONENTES.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes *software*, sean éstos componentes de código fuente, binarios o ejecutables. Como todos los diagramas, también puede contener paquetes utilizados para agrupar elementos del modelo. [24]

La vista de componentes muestra los componentes integrados:

Se tiene una forma *Principal* la cual se va a conectar con la forma *Gestionar Paciente*, que es donde se *Busca el Paciente*, si este no se encuentra, se va a la forma *Adicionar Paciente* quien después de realizar dicha función se conecta con la clase *Save,* esta contiene *EntityManager* quien se encarga de persistir los datos y adicionarlos a la *Base de Datos*.

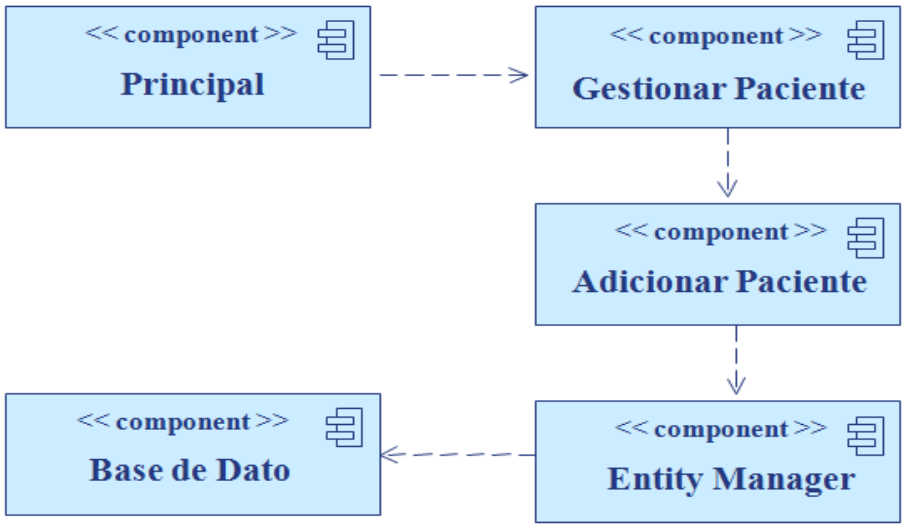


Fig. 6 Vista de Componentes.

2.8. ESTÁNDARES DE CODIFICACIÓN.

Cuando se trabaja en equipo es necesario hacer un código legible y entendible, no sólo para quien lo escribe, sino también para quien lo lee, y para eso es necesario tener en cuenta varios aspectos:

- Las cláusulas, la notación que se utilizará para nombrar cada uno de los identificadores que se declaran.
- La estructura del código en sí, referente a las tabulaciones y los espacios entre líneas y dentro de las líneas, los espacios entre los operadores y estructuras que componen el lenguaje en que se desarrolla la aplicación.
- Usar siempre rutinas de manejo de excepciones.
- El nombre de la clase y el archivo fuente deben ser iguales.
- Los comentarios deben ser en español.

El uso de los estándares de codificación de dicha investigación está presente de la siguiente manera:

2.8.1- NOTACIÓN CAMELLO.

Se usa para denotar variables y parámetros. En esta notación, si el identificador es una palabra simple se escribe todo con minúscula, pero si es compuesta, la primera letra de todas las palabras que viene a continuación de la primera comienza con mayúscula.

Ejemplo:

```
    private Integer id;

    private PersonalMedico personalMedicoid;

   private Date fechaConsulta;

 public void setPacientehc (Paciente pacientehc)
{
    this.pacientehc = pacientehc;
}
```

2.8.2. MÉTODOS.

Los nombres de métodos deben describir la acción que realizan, y si el identificador es compuesto, la primera palabra debe ser el infinitivo de la acción. Ejemplo:

```
    public List<Paciente> BuscarPaciente ()
  {
      //…
  }
```

2.8.3. PROPIEDADES (Properties)

Si modifican o devuelven algún atributo perteneciente a una clase, debe tener el mismo nombre del atributo, pero su primera letra debe ser mayúscula. De otra forma deben seguir las cláusulas de los métodos.

Ejemplo:

```
public String getNombre ()
{
        return nombre;
}
public void setNombre (String nombre)
{
    this.nombre = nombre;
}
```

Como se pudo observar los estilos de código hacen que el programa fuente sea más legible, lo cual ayuda en la comunicación entre desarrolladores, y permite una temprana detección de errores. La propuesta es completamente extensible.

Siempre existe un riesgo cuando se definen estilos de codificación en aplicar más prefijos o sufijos en la sintaxis a conceptos que ya tienen una forma de ser expresados, cayendo en una redundancia expresiva.

2.9. ENTIDADES DEL NEGOCIO.

Estas entidades fueron creadas siguiendo el patrón de mapeo de datos, que propone crear un objeto por cada entidad persistente (tabla de la BD).

Nombre: Unidad_ Salud	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Unidad_Salud
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el nombre.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el id.

Tabla 1 Clase Entidad "*Unidad_Salud*".

Nombre: Paciente	
Tipo de clase: Entidad	
Atributo	**Tipo**
hc	Int32
nombre	String
primer_apellido	String
segundo_apellido	String
sexo	char
grupo	Int32
carnet_ID	Int32
direccion_particular	String

edad	Int32
Para cada responsabilidad:	
Nombre:	Paciente
Descripción:	Constructor de la clase.
Nombre:	hc
Descripción:	Propiedad para mostrar y modificar la hc.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.
Nombre:	primer_apellido
Descripción:	Propiedad para mostrar y modificar el primer_apellido.
Nombre:	segundo_apellido
Descripción:	Propiedad para mostrar y modificar el segundo_apellido.
Nombre:	sexo
Descripción:	Propiedad para mostrar y modificar el sexo.
Nombre:	grupo
Descripción:	Propiedad para mostrar y modificar el grupo.
Nombre:	carnet_ID
Descripción:	Propiedad para mostrar y modificar el carnet_ID.
Nombre:	direccion_particular
Descripción:	Propiedad para mostrar y modificar la direccion_particular.
Nombre:	edad
Descripción:	Propiedad para mostrar y modificar la edad.

Tabla 2 Clase Entidad *"Paciente"*.

Nombre: Usuario	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
usuario	String
contraseña	String
Para cada responsabilidad:	
Nombre:	usuario
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	usuario
Descripción:	Propiedad para mostrar y modificar el usuario.
Nombre:	contraseña
Descripción:	Propiedad para mostrar y modificar la contraseña.

Tabla 3 Clase Entidad *"Usuario"*.

Nombre: Sistema Médico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Sistema Médico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 4 Clase Entidad *"Sistema Médico"*.

Nombre: Sistema Ginecológico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
edad_menarquía	Int32
edad_menopausia	Int32
fecha_primeraRelacionsexual	date
fecha_ultimaMestruacion	date
ciclo_mestrual	String
partos	Int32
abortos	Int32
macrofetos	Int32
tiempo_uso_anticonceptivo	String
fecha_prueba_citológica	date
resultados	String
Para cada responsabilidad:	
Nombre:	Sistema Ginecológico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	edad_menarquía
Descripción:	Propiedad para mostrar y modificar la edad_menarquía.
Nombre:	edad_menopausia

Descripción:	Propiedad para mostrar y modificar la edad_menopausia.
Nombre:	fecha_primeraRelacionsexual
Descripción:	Propiedad para mostrar y modificar la fecha_primeraRelacionsexual.
Nombre:	fecha_ultimaMestruacion
Descripción:	Propiedad para mostrar y modificar la fecha_ultimaMestruacion.
Nombre:	ciclo_mestrual
Descripción:	Propiedad para mostrar y modificar el ciclo_mestrual.
Nombre:	partos
Descripción:	Propiedad para mostrar y modificar el número de partos.
Nombre:	abortos
Descripción:	Propiedad para mostrar y modificar si ha tenido abortos.
Nombre:	macrofetos
Descripción:	Propiedad para mostrar y modificar el macrofeto.
Nombre:	tiempo_uso_anticonceptivo
Descripción:	Propiedad para mostrar y modificar el tiempo_uso_anticonceptivo.
Nombre:	fecha_prueba_citológica
Descripción:	Propiedad para mostrar y modificar la fecha_prueba_citológica
Nombre:	resultados
Descripción:	Propiedad para mostrar y modificar los resultados.

Tabla 5 Clase Entidad *"Sistema Ginecológico".*

Nombre: Intervenciones Quirúrgicas	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
fecha_intervención	date
secuelas	String
Para cada responsabilidad:	
Nombre:	Intervenciones Quirúrgicas
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fecha_intervención
Descripción:	Propiedad para mostrar y modificar la fecha_intervención.
Nombre:	secuelas
Descripción:	Propiedad para mostrar y modificar las secuelas.

Tabla 6 Clase Entidad *"Intervenciones Quirúrgicas".*

Nombre: Tipo de Intervenciones	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Tipo de Intervenciones
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 7 Clase Entidad *"Tipo de Intervenciones"*.

Nombre: Historial de Enfermedades	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
Para cada responsabilidad:	
Nombre:	Historial de Enfermedades
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Tabla 8 Clase Entidad *"Historial de Enfermedades"*.

Nombre: Enfermedades	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Enfermedades
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre

Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 9 Clase Entidad *"Enfermedades".*

Nombre: Frecuencia	
Tipo de clase: Entidad	
Atributo	**Tipo**
Id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Frecuencia
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 10 Clase Entidad *"Frecuencia".*

Nombre: Historia Clínica	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
Para cada responsabilidad:	
Nombre:	Historia Clínica
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Tabla 11 Clase Entidad *"Historia Clínica".*

Nombre: Tratamiento	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Tratamiento
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 12 Clase Entidad *"Tratamiento".*

Nombre: Síntomas	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Síntomas
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre

Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 13 Clase Entidad *"Síntomas"*.

Nombre: Proceder de Enfermería	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
fecha_proceder	date
descripción	String
Para cada responsabilidad:	
Nombre:	Proceder de Enfermería
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fecha_proceder
Descripción:	Propiedad para mostrar y modificar la fecha_proceder.
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.
Nombre:	
Descripción:	Propiedad para mostrar y modificar el.

Tabla 14 Clase Entidad *"Proceder de Enfermería"*.

Nombre: Certificado Médico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32

fecha_certificado	date
días_incapacidad	int
descripción	String
Para cada responsabilidad:	
Nombre:	Certificado Médico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fecha_certificado
Descripción:	Propiedad para mostrar y modificar la fecha_certificado.
Nombre:	días_incapacidad
Descripción:	Propiedad para mostrar y modificar los días_incapacidad.
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 15 Clase Entidad *"Certificado Médico"*.

Nombre: Certificado Tarjeta	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
dosis_diaria	String
día_orientado	date
descripción	String
Para cada responsabilidad:	
Nombre:	Certificado Tarjeta
Descripción:	Constructor de la clase.
Nombre:	id

Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	dosis_diaria
Descripción:	Propiedad para mostrar y modificar el dosis_diaria.
Nombre:	día_orientado
Descripción:	Propiedad para mostrar y modificar el día_orientado.
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 16 Clase Entidad *"Certificado Tarjeta".*

Nombre: Examen Físico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
Para cada responsabilidad:	
Nombre:	Examen Físico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Tabla 17 Clase Entidad *"Examen Físico".*

Nombre: Descripción de Examen Físico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
descripción	String
Para cada responsabilidad:	

Nombre:	Descripción de Examen Físico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 18 Clase Entidad *"Descripción de Examen Físico"*.

Nombre: Medicamento	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
descripción	String
código	String
Para cada responsabilidad:	
Nombre:	Medicamento
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.
Nombre:	código
Descripción:	Propiedad para mostrar y modificar el código.

Tabla 19 Clase Entidad *"Medicamento"*.

Nombre: Método	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	int
unidad_medida	double
descripción	String
Para cada responsabilidad:	
Nombre:	Método
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id
Nombre:	unidad_medida
Descripción:	Propiedad para mostrar y modificar la unidad_medida.
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 20 Clase Entidad *"Método"*.

Nombre: Pedido	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
unidad_medida	double
cantidad_orden	Int32
despacho	Int32
tipo_orden	String
día_pedido	date
Para cada responsabilidad:	

Nombre:	Pedido
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	unidad_medida
Descripción:	Propiedad para mostrar y modificar la unidad_medida
Nombre:	cantidad_orden
Descripción:	Propiedad para mostrar y modificar la cantidad_orden.
Nombre:	despacho
Descripción:	Propiedad para mostrar y modificar el despacho.
Nombre:	tipo_orden
Descripción:	Propiedad para mostrar y modificar el tipo_orden.
Nombre:	día_pedido
Descripción:	Propiedad para mostrar y modificar el día_pedido.

Tabla 21 Clase Entidad *"Pedido"*.

Nombre: Presentación	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Presentación
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre

Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 22 Clase Entidad *"Presentación"*.

Nombre: Producto	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Producto
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 23 Clase Entidad *"Producto"*.

Nombre: Hábitos Tóxicos	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Hábitos Tóxicos
Descripción:	Constructor de la clase.
Nombre:	id

Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 24 Clase Entidad *"Hábitos Tóxicos"*.

Nombre: Alcoholismo	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
Para cada responsabilidad:	
Nombre:	Alcoholismo
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Tabla 25 Clase Entidad *"Alcoholismo"*.

Nombre: Otros	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
descripción	String
Para cada responsabilidad:	
Nombre:	Otros
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 26 Clase Entidad *"Otros"*.

Nombre: Tabaquismo	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
fumador	String
exfumador	String
cant_fuma	Int32
tiempo_fumando	String
fuma_tabaco	String
fuma_cigarro	String
Para cada responsabilidad:	
Nombre:	Tabaquismo
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fumador
Descripción:	Propiedad para mostrar y modificar el fumador.
Nombre:	exfumador
Descripción:	Propiedad para mostrar y modificar el exfumador.
Nombre:	cant_fuma
Descripción:	Propiedad para mostrar y modificar la cant_fuma.
Nombre:	tiempo_fumando
Descripción:	Propiedad para mostrar y modificar el tiempo_fumando.

Nombre:	fuma_tabaco
Descripción:	Propiedad para mostrar y modificar fuma_tabaco.
Nombre:	fuma_cigarro
Descripción:	Propiedad para mostrar y modificar fuma_cigarro.

Tabla 27 Clase Entidad *"Tabaquismo"*.

Nombre: Remisión Especialista	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
fecha_remisión	date
descripcion	String
Para cada responsabilidad:	
Nombre:	Remisión Especialista
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fecha_remisión
Descripción:	Propiedad para mostrar y modificar el fecha_remisión.
Nombre:	descripción
Descripción:	Propiedad para mostrar y modificar la descripción.

Tabla 28 Clase Entidad *"Remisión Especialista"*.

Nombre: Complementarios	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
día_mandado	date
día_ realizado	date
resultados	String
Para cada responsabilidad:	
Nombre:	Complementarios
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	día _mandado
Descripción:	Propiedad para mostrar y modificar el día mandado.
Nombre:	día_realizado
Descripción:	Propiedad para mostrar y modificar el día_realizado.
Nombre:	resultados
Descripción:	Propiedad para mostrar y modificar los resultados.

Tabla 29 Clase Entidad *"Complementarios"*.

Nombre: Tipo Complementario	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Tipo Complementario

Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 30 Clase Entidad *"Tipo Complementarios"*.

Nombre: Especialidades	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String
Para cada responsabilidad:	
Nombre:	Especialidades
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 31 Clase Entidad *"Especialidades"*.

Nombre: Anticonceptivo	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String

Para cada responsabilidad:	
Nombre:	Anticonceptivo
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 32 Clase Entidad *"Anticonceptivo"*.

Nombre: Consulta	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
fecha_consulta	date
diagnóstico	String
Para cada responsabilidad:	
Nombre:	Consulta
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	fecha_consulta
Descripción:	Propiedad para mostrar y modificar la fecha_consulta.
Nombre:	diagnóstico

Tabla 33 Clase Entidad *"Consulta"*.

Nombre: Parentesco	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	String
nombre	Int32
Para cada responsabilidad:	
Nombre:	Parentesco
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.

Tabla 34 Clase Entidad *"Parentesco"*.

Nombre: Personal Médico	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
nombre	String,
primer_apellido	String
segundo_apellido	String
especialidad	String
Para cada responsabilidad:	
Nombre:	Personal Médico
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.

Nombre:	nombre
Descripción:	Propiedad para mostrar y modificar el nombre.
Nombre:	primer_apellido
Descripción:	Propiedad para mostrar y modificar el primer_apellido.
Nombre:	segundo_apellido
Descripción:	Propiedad para mostrar y modificar el segundo_apellido.
Nombre:	especialidad
Descripción:	Propiedad para mostrar y modificar la especialidad.

Tabla 35 Clase Entidad *"Personal Médico"*.

Nombre: Receta Médica	
Tipo de clase: Entidad	
Atributo	**Tipo**
id	Int32
unidad_medida	double
Para cada responsabilidad:	
Nombre:	Receta Médica
Descripción:	Constructor de la clase.
Nombre:	id
Descripción:	Propiedad para mostrar y modificar el id.
Nombre:	unidad_medida
Descripción:	Propiedad para mostrar y modificar el unidad_medida.

Tabla 36 Clase Entidad *"Receta Médica"*.

CONCLUSIONES

Este capítulo es uno de los más importantes de la investigación, puesto que se describe como está implementado el sistema, los módulos con los que necesita integrarse para su óptimo funcionamiento. Además se brindó una descripción detallada de las clases que conforman la solución del problema, facilitando su entendimiento para futuras actualizaciones.

CAPÍTULO III
VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

Uno de los mayores problemas que se afrontan actualmente en la esfera de la informática es la calidad del software, por lo que el proceso de pruebas es sin dudas uno de los aspectos fundamentales para medir el estado de calidad de un sistema informático.

La obtención de un software con calidad, implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software, que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad de la aplicación. A raíz de la incapacidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software debe ser acompañado de una actividad que garantice su calidad. [25]

En el desarrollo del *software*, las posibilidades de errores son innumerables. Pueden darse por una mala especificación de los requisitos funcionales, uso indebido de las estructuras de datos, errores al enlazar módulos, etcétera.

La prueba y validación de los resultados no es un proceso que se realiza una vez desarrollado el software, sino que debe efectuarse en cada una de las etapas de desarrollo. Es fundamental medir la cobertura de las pruebas, es decir, la determinación de cuando se han realizado las suficientes pruebas. Si se siguen encontrando errores cada vez que se procesa el programa, las pruebas deben continuar.

Durante el mantenimiento debe existir una documentación de pruebas que incluya casos de prueba y resultados esperados. Si se producen modificaciones en el programa, habrá que probar de nuevo todas las partes del programa afectadas por las modificaciones.

Por lo dicho anteriormente, se desarrolló este capítulo, con el objetivo de especificar las pruebas que serán hechas al software y describir todo lo encontrado en las mismas.

3.1. PRUEBAS APLICADAS.

Las pruebas son la actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, donde los resultados son observados y registrados, y es hecha una evaluación de algún aspecto del sistema o componente.

Con la realización de estas pruebas se pretende encontrar y documentar los defectos que puedan afectar la calidad del *software*, validar y probar los requisitos que debe cumplir el *software* y a su vez que estos fueron implementados correctamente.

Es necesario analizar que las pruebas no pueden asegurar la ausencia de defectos sino que permiten demostrar que existen defectos en el *software*, que cada prototipo que se quiera entregar al final de una iteración debe ser probado y evaluado.

Los casos de prueba especifican la forma de probar el sistema en cuestión, incluyendo la entrada o resultado con el que se ha de probar y las condiciones bajo las que ha de probarse. Es un conjunto de entradas y resultados esperados que ponen en práctica el funcionamiento de un componente con el objetivo de causar fallas y detectar defectos. Entre los casos de prueba pueden existir todos los tipos de relaciones, pero las más importantes son las de precedencia.

Todo producto puede ser probado de las siguientes formas:

1. Conociendo el funcionamiento del producto, para poder desarrollar pruebas que aseguren que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

2. Conociendo la funcionalidad específica para la cual fue diseñado el producto, para poder llevar a cabo pruebas que demuestren que cada función es completamente operativa.

En el primer caso, las pruebas están dirigidas a la aplicación de métodos basados en Caja Blanca y en el segundo caso, a los métodos basados en Caja Negra.

Los de este primer grupo permiten la comprobación de los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones.

3.2. TIPOS DE PRUEBA

- Pruebas de caja blanca.
- Pruebas de caja negra.

3.2.1. PRUEBAS DE CAJA BLANCA.

Las pruebas de caja blanca normalmente se denominan pruebas de cobertura o pruebas de caja transparente. Al total de pruebas de caja blanca se le llama cobertura. La cobertura es un número porcentual que indica cuanto código del programa se ha probado.

Básicamente la idea de pruebas de cobertura consiste en diseñar un plan de pruebas en las que se vaya ejecutando sistemáticamente el código hasta que haya corrido todo o la gran mayoría de él, esto que parece complicado, pero lo es más, cuando el programa contiene código de difícil alcance, como por ejemplo manejadores de errores o "código muerto".

Es muy recomendable alcanzar una elevada cobertura de sentencias, aunque no siempre es posible por premura de tiempo o medios. La prueba de caja blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la caja blanca se puede obtener casos de prueba que:

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

3.2.1.1. DESCRIPCIÓN DE LAS PRUEBAS DE CAJA BLANCA REALIZADAS.

Para llevar a cabo los casos de pruebas de caja blanca previstas en el epígrafe anterior, fue preciso tomar muestra del código fuente de la aplicación, para esto se tuvo en cuenta la implementación del método "*Buscar IntervencionesQuirúrgicas*".

En este caso se determinó el grafo de flujo y por ende la complejidad ciclomática, obteniéndose la cantidad de caminos independientes necesarios para aplicar los casos de pruebas correspondientes a cada una.

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

- Porción de código para buscar las intervenciones quirúrgicas:

```
public List<IntervencionesQuirurgicas> Buscar ()  // es el método encargado de buscar por parámetros
    {
        EntityManagerFactory pac = Persistence.createEntityManagerFactory ("tesisPU"); (1)
        EntityManager em = pac.createEntityManager ();    (1)
        Listpa = em.createQuery ("SELECT p FROM IntervencionesQuirurgicas p where
p.historiaClinicaid.pacientehc.hc =: hc").setParameter ("hc", identif).getResultList ();    (1)
          If (listpa.size () > 0)      (2)
        {
```

```
        s = new String [listpa.size ()] [7]; (2)
        int i = 0;            (2)
        for (IntervencionesQuirurgicas p: listpa)    (3)
    {
            String [] str = {p.getTipoIntervencionesid ().getNombre (), p.getFechaIntervencion
().toString (), p.getSecuelas ()};        (3)
            s[i] = str;       (3)
            i++;          (4)
        }
                } (5)
        else
        {
          JOptionPane.showMessageDialog (null, "No tiene ninguna intervención quirúrgicas"); (6)
        }
          return listpa;              (7)
       }     (7)
```

❖ Construcción del grafo de flujo asociado al código representado anteriormente:

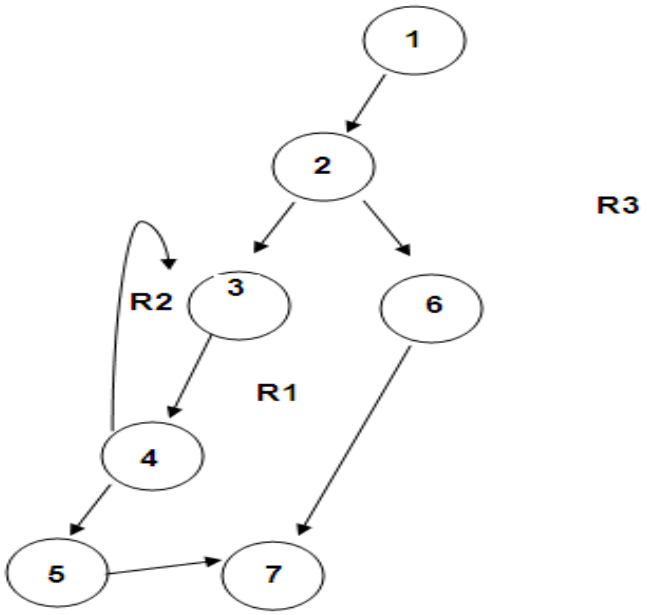


Fig. 7 Representación del grafo del método "*Buscar Intervenciones Quirúrgicas*".

Aplicación de las fórmulas para calcular complejidad ciclomática. Para que el resultado este correcto debe ser el mismo para las tres fórmulas.

- V (G) = (A – N) + 2.
- V (G) = P + 1.
- V (G) = R.

En este primer caso se obtuvo que:

- V (G) = 8 aristas - 7 nodos + 2 = 3.
- V (G) = 2 nodos predicados + 1 = 3.
- V (G) = 3 regiones.

Por tanto la complejidad ciclomática del grafo de flujo de la figura es igual a 3, esto significa que existen tres posibles caminos por donde el flujo puede circular.

Como caminos independientes se determinaron:

Camino 1: 1- 2 - 3 - 4 - 5 - 7.

Camino 2: 1 - 6 - 7.

Camino 3: 1- 2 - 3 - 4 - 3 - 4 - 5 - 7.

Este algoritmo está conformado por varias sentencias repetitivas (bucles) que se ejecutan una tras la otra, de esta manera solo es necesario probar que estos bucles se ejecutan completamente, garantizando una cobertura total con una prueba, donde se introduzca la información necesaria para que todos sean ejecutados. El camino básico que será objetivo de prueba es: 1-2-3-4-5-7.

Para este caminos independientes se conformaron el siguiente caso de prueba:

Caso de prueba para el Camino 1:

Descripción y asignación:

Se quiere determinar si el paciente se ha realizado alguna intervención quirúrgica, esto se debe encontrar registrado en la historia clínica del paciente, para esto es asignada la variable *Listpa,* que es quien va a retornar la lista de las intervenciones realizadas.

Condición de ejecución:

Se especifica cada parámetro para que cumpla la condición deseada y ver el funcionamiento del procedimiento. Todos los datos deben ser entrados.

Entrada:

Datos generales:

Id = 1, fecha_intervención = 16/06/2008, secuelas =.

Resultado esperado:

Al entrar todos los datos al sistema, se muestran todas las intervenciones quirúrgicas que se ha realizado el paciente, especificando el id, la fecha y las secuelas que le han quedado al paciente luego de haber sido intervenido.

- Evaluación de los resultados obtenidos.

Para el caso de prueba descrito, el algoritmo recorrió todos los bucles, confirmando su buen funcionamiento mostrando los resultados esperados.

Luego de aplicar el casos de pruebas, se pudo comprobar que el flujo de trabajo del procedimiento está correcto ya que cumple con las condiciones necesarias que se habían planteado para este procedimiento.

3.2.2. PRUEBAS DE 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.

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.

3.2.2.1. DESCRIPCIÓN DE LAS PRUEBAS DE CAJA NEGRA REALIZADAS.

Para la aplicación de este tipo de prueba se usará el caso de uso "*Gestionar Pedidos*".

Objetivo del test:

Validar el caso de uso "*Gestionar Pedidos*".

Las pruebas realizadas a este caso de uso son:

- Adicionar *Pedido*.
- Buscar *Pedido.*
- Despachar *Pedido*.
- Eliminar *Pedido*.

Flujo central:

Una vez que el consultorio presenta déficit de materiales, la enfermera hace una solicitud al almacén con un pedido de los mismos. En dicho pedido se especifica el tipo de pedido que se desea, el consultorio al que pertenece, el producto o medicamento que se solicita, la fecha y la cantidad. Una vez llenado todos los datos la enfermera pulsa el botón "*aceptar"*.

Pre condiciones:

Tienen que estar presente en la base de datos:
- Pedidos.

- Medicamentos.
- Producto.
- Unidad de salud.

Sección: Buscar Pedido					
Clases válidas	**Clases no válidas**	**Resultados de la prueba.**	**Resultados de la prueba.**	**Observaciones**	**Cumplimento**
El usuario selecciona la opción “Gestionar Pedido” del menú principal de trabajo.	El sistema no muestra la interfaz correspondiente a Pedido.	El sistema muestra la interfaz correspondiente a Pedido.	Satisfactoria	La operación se realizó correctamente.	100%
El usuario llena los datos necesarios y presiona el botón “Buscar”.	No se muestra la interfaz de “*Buscar Pedidos*”.	El sistema muestra aquellos pedidos que coincidan con los datos entrados.	Satisfactoria	Si no se pudo registrar la información el sistema muestra el siguiente mensaje de error: “**Error: "No existen pedidos con esos parámetros.”**	100%

El usuario accede al Pedido deseado.	El sistema no muestra el pedido.	El sistema muestra el pedido buscado.	Satisfactoria	El sistema muestra el pedido, y haciendo uso de la misma interfaz adiciona y despacha dichos pedidos.	100%

Tabla 2. "Sección: *Buscar Pedido*".

Sección: Adicionar Pedido.					
Clases válidas	**Clases no válidas**	**Resultados de la prueba.**	**Resultados de la prueba.**	**Observaciones**	**Cumplimiento**
La enfermera selecciona la opción "Gestionar Pedido" del menú principal de trabajo.	No se muestra la interfaz del Pedido.	Muestra la interfaz del Pedido.	Satisfactoria	Haciendo uso de la interfaz "buscar pedido" adiciona.	100%
La enfermera selecciona la opción *"Adicionar".*	No se muestra la interfaz "Adicionar Pedido"	Muestra la interfaz "Adicionar Pedido"	Satisfactoria	La operación se realizó correctamente.	100%

La enfermera recoge los datos específicos de los pedidos y presiona el botón "*Aceptar*".	No se muestra la interfaz de "*Adicionar Pedidos*".	Muestra la interfaz "*Adicionar Pedidos*".	Satisfactoria	Si no se pudo registrar la información el sistema muestra el siguiente mensaje de error: "**Error: "Faltan datos obligatorios por entrar."**	100%

Tabla 3. Sección:"*Adicionar Pedido*".

Sección: Despachar Pedido.					
Clases válidas	**Clases no válidas**	**Resultados de la prueba.**	**Resultados de la prueba.**	**Observaciones**	**Cumplimiento**
La enfermera selecciona la opción "Gestionar Pedido" del menú principal de trabajo.	No se muestra la interfaz del Pedido.	Muestra la interfaz del Pedido.	Satisfactoria	Haciendo uso de la interfaz "buscar pedido" despacha.	100%
La enfermera realiza la búsqueda del	No se muestra el pedido buscado.	Muestra el pedido buscado.	Satisfactoria	La operación se realizó correctamente.	100%

pedido.					
La enfermera selecciona la opción *"Despachar".*	No se muestra la interfaz "Despachar Pedido"	Muestra la interfaz "Despachar Pedido"	Satisfactoria	Una vez que aparece el pedido buscado, se despacha.	100%
La enfermera llena los datos específicos de los pedidos y presiona el botón "*Aceptar*".	No se muestra la interfaz de "*Modificar Pedidos*".	Muestra la interfaz "*Modificar Pedidos*".	Satisfactoria	Si no se pudo registrar la información el sistema muestra el siguiente mensaje de error: "**Error: "Faltan datos obligatorios por entrar."**	100%

Tabla 4. Sección:"*Despachar Pedido*".

Sección: Eliminar Pedido.					
Clases válidas	**Clases no válidas**	**Resultados de la prueba.**	**Resultados de la prueba.**	**Observaciones**	**Cumplimiento**
La enfermera selecciona la opción "Gestionar Pedido" del	No se muestra la interfaz del Pedido.	Muestra la interfaz del Pedido.	Satisfactoria	Haciendo uso de la interfaz "buscar pedido" elimina.	100%

menú principal de trabajo.					
La enfermera realiza la búsqueda del pedido.	No se muestra el pedido buscado.	Muestra el pedido buscado.	Satisfactoria	La operación se realizó correctamente.	100%
La enfermera selecciona el pedido y da clic en la opción *"Eliminar".*	No se muestra la interfaz "Eliminar Pedido"	Muestra la interfaz "Eliminar Pedido"	Satisfactoria	La operación se realizó correctamente.	100%

Tabla 5. Sección:"*Eliminar Pedido*".

CONCLUSIONES

En este capítulo se describen los métodos de pruebas usados en el desarrollo de la aplicación. Además se realizaron las pruebas de caja blanca y caja negra, demostrando su adecuado funcionamiento y reflejando la calidad con que ha sido llevada a cabo la proyección del sistema. Estas acciones realizadas durante la implementación, tienen el objetivo de asegurar que el sistema implementado tenga la calidad necesaria para lograr la satisfacción del cliente.

CONCLUSIONES

En el presente trabajo de diploma quedó demostrada la necesidad de implementar un sistema informático en ambiente desconectado, que permita agilizar la gestión de la información médica de los pacientes en el consultorio de la facultad 7 de la Universidad de las Ciencias Informáticas.

Con el desarrollo de esta investigación se arribó a las siguientes conclusiones:

- Se realizó un estudio sobre las principales problemáticas existentes actualmente en los consultorios para darle solución a las mismas.
- Se llevó a cabo una profunda investigación de los lenguajes y tecnologías de desarrollo.
- Para brindar una adecuada seguridad en el almacenamiento de la información se realizó el diseño de las clases y de la base de datos y junto con ellos se seleccionó el framework adecuado para facilitar el acceso y la persistencia de los mismos.

De esta forma se le da cumplimiento a los objetivos trazados para la elaboración del trabajo de diploma, obteniendo la implementación de un sistema informático en ambiente desconectado que permitirá darle solución a la mayoría de los procesos que se gestionan en los consultorios médicos.

RECOMENDACIONES

Por la experiencia adquirida con este trabajo los autores recomiendan:

- ✓ Lograr una mayor profundización del tema abordado, para ampliar los conocimientos con vista a detectar posibles debilidades en la implementación del sistema.
- ✓ Continuar promoviendo la vinculación temprana, por parte de los estudiantes de la Universidad y en especial de la facultad, a proyectos productivos que ayuden en la informatización de la salud.
- ✓ Confeccionar un sistema de ayuda que permita comprender de forma fácil y entendible el manejo de la presente aplicación.
- ✓ Desplegar el sistema hacia todos los consultorios de la universidad y posteriormente a todo el país.
- ✓ Lograr en un futuro la implementación de una aplicación o sistema Web que brinde las funcionalidades existentes y las que podrían aparecer a lo largo de la prueba piloto de la presente versión realizada como aplicación de escritorio.

REFERENCIAS BIBLIOGRÁFICAS

[1]. Alcides, Lorenzo Rodriguez**.**. *ENSAP.* [En línea] 2003. [Citado el: 15 de enero del 2008.] Disponible en: http://www.sld.cu/galerias/doc/sitios/infodir/19_lorenzo_rodriguez_alcides.doc.

[2]. Ídem al [1].

[3]. *Visual Medic, Sistema Integral de Archivo Clínico.* [En línea] [Citado el: 05 de febrero del 2008.] Disponible en: http://www.visualmedic.net/vminfo.htm .

[4]. *Atención Primaria de la Salud.* [En línea] [Citado el: 30 de enero del 2008.] Disponible en: http://aps.sld.cu/bvs/E/sobreaps.html.

[5]. *Intermedical.freeservers.com.* [En línea] [Citado el: 20 de febrero del 2008.] Disponible en : http://intermedical.freeservers.com/descripcion%20resumida%20intermedical%20soft.html.

[6]. *JagarSoft.* [En línea] 2002. [Citado el: 20 de febrero del 2008.] Disponible en: http://www.jagarsoft.com/.

[7]. Morales, Dr. Juan Carlos García**.** *SIDAPS: SISTEMA INFORMÁTICO PARA LA DISPENSARIZACIÓN EN LA ATENCIÓN PRIMARIA DE SALUD.* [En línea] [Citado el: 06 de marzo del 2008.] Disponible en: http://www.cecam.sld.cu/pages/rcim/revista_15/articulos_htm/sidaps.htm.

[8]. Fernández, Giovanny. *EstándarCodificación DOTNET.* [En línea] 21 de abril de 2005. [Citado el: 16 de marzo del 2008.] Disponible en: http://www.elguille.info/colabora/NET2005/giovannyfernandez_EstandarCodificacionNET.htm.

[9]. Ídem al [8].

[10]. Ídem al [8].

[11]. Seco, José Antonio González**.** *devjoker,Tutorial de C#.* [En línea] 03 de octubre del 2006. [Citado el: 10 de abril del 2008.] Disponible en: http://www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=125#a1.

[12]. Ídem al [11].

[13]. Froufe, Agustín**.** *Itapizaco,Tutorial de Java.* [En línea] 17 de mayo de 1999. [Citado el: 23 de abril del 2008.] Dipsonible en: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte2/cap2-5.html.

[14]. Laura Bermejo Sanz, Enrique Gómez Monreal**.** *Eclipse como IDE.* [En línea] [Citado el: 03 de mayo del 2008.] http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf.

[15]. Ramírez, Iván**.** *Softnic, NetBeans IDE.* [En línea] 1997. [Citado el: 03 de mayo del 2008.] Disponible en: http://netbeans-ide.softonic.com/.

[16]. *NetBeans.* [En línea] [Citado el: 05 de mayo del 2008.] Disponible en: http://www.netbeans.org/index_es.html.

[17]. *itapizaco,Tutorial de Java, Swing.* [En línea] [Citado el: 05 de mayo del 2008.]Disponible en: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte14/cap14-1.html .

[18]. Pozo, Salvador. *MySQL con Clase.* [En línea] marzo del 2004. [Citado el: 06 de mayo del 2008.] Disponible en: http://mysql.conclase.net/curso/index.php.

[19]. Ídem [18].

[20]. Pecos, Daniel**.** *PostgreSQL vs. MySQL.* [En línea] [Citado el: 08 de mayo del 2008.] Disponible en: http://www.netpecos.org/docs/mysql_postgres/x15.html.

[21]. Hispano, Asociación Java**.** *Java Hispano.* [En línea] de de 2002. [Citado el: 14 de mayo de 2008.] Disponible en: http://www.javahispano.org/contenidos/es/manual_hibernate/.

[22]. Rizzi, Ing. Francisco Marcelo**.** *Itba, Complejidad Ciclomática.* [En línea] [Citado el: 20 de mayo del 2008.] Disponible en: http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZI-COMPLEJIDAD.pdf .

[23]. Vilas, Ana Fernández. *Diagrama de Componente.* [En línea] 20 de marzo del 2001. [Citado el: 05 de junio del 2008.] Disponible en: http://www-gris.det.uvigo.es/~avilas/UML/node49.html.

[24]. Ídem al [23].

[25]. Oscar M. Fernández Carrasco, Delba García León, Alfa Beltrán Benavides**.** Enfoque Actual sobre la Calidad del Software. [En línea] septiembre - diciembre de 1995. [Citado el: 10 de junio del 2008.] Disponible en: http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.

BIBLIOGRAFÍA

1. Alcides Lorenzo Rodriguez . ENSAP. Publicado en el 2003. Consultado el 15 de enero del 2008. Disponible en: http://www.sld.cu/galerias/doc/sitios/infodir/19_lorenzo_rodriguez_alcides.doc.

2. Atención Primaria de la Salud. Atención Primaria de la Salud. Consultado el 30 de enero del 2008. Disponible en: http://aps.sld.cu/bvs/E/sobreaps.html.

3. Athos A Sánchez Mansolo, Jorge Luis Iglesias Dios, Gabriel Perdomo González, José, Dayma Mendoza. Revista Cubana de Informática Médica . ELECTRONIC CLINICAL RECORDS IN CUBA, A DREAM OR A REAL POSSIBILITY. Publicado en el 2000. Disponible en: http://www.cecam.sld.cu/pages/rcim/revista_1/articulos_pdf/r0100a05.pdf.

4. AurumSolution .Publicado en noviembre del 2003. Disponible en: http://espanol.aurumsol.com/articulos/art6/art6-print.html.

5. CARTER, J. Software Engineering Institute. Community Software Architecture Definitions. Publicado el 23 de marzo del 2007.

6. Concepto y características de los SGBD publicado el 29 de marzo del 2007.. - MADRID, U. C. D., 1997. Disponible en: http://www.eubd.ucm.es/html/personales/enred/mantonia/docauto/tema5/tema5.htm.

7. DATAHOUSE COMPANY- Software para Consultorios Médicos. Disponible en: http://www.datahousecompany.com.ar/consultorios-medicos.html.

8. DeremaTe.Publicado en 1999. última actualización en el 2007. Disponible en: http://oferta.deremate.com.ar/id=18331768_software-consultorio-médico.

9. Diseño de Base de Datos. Conferencias. Teleformación.uci.cu. última actualización en el 2008. - http://teleformacion.uci.cu/mod/resource/view.php?id=22077.

10. Dr. Pedro Luis Monteagudo Valdivia MSc, Lic. Alfredo Rodríguez Díaz, Dra. Maylid Hernández Medina INFORMEDICA Publicado en el 2004. Disponible en: http://www.informaticamedica.org/I04/papers/monteagudovaldivia_43.pdf.

11. Enfermeria Global- QUALITY IN THE HEALTH PRIMARY ATTENTION. Autora Rosa D. Villalba, publicado en mayo del 2007. Disponible en: http://www.um.es/ojs/index.php/eglobal/article/viewFile/199/168.

12. Englander Robert Java Beans. [Libro]. Publicado en 1997.

13. Fernández Giovanny EstandarCodificación DOTNET .EstándarCodificación DOTNET. última actualización 21 de abril de 2005. Consultado el 16 de marzo del 2008. Disponible en: http://www.elguille.info/colabora/NET2005/giovannyfernandez_EstandarCodificacionNET.htm.

14. Flanagan. David Java en Pocas Palabras. [Libro]. Publicado en 1999. 2da Edición.

15. Flujo de Trabajo de Pruebas . Conferencias. Teleformación.uci.cu. Disponible en: http://teleformacion.uci.cu/mod/resource/view.php?id=22393.

16. Froufe Agustín Itapizaco,Tutorial de Java. Itapizaco,Tutorial de Java. Publicado el 17 de mayo de 1999. Consultado el 23 de abril del 2008.Disponible en: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte2/cap2-5.html.

17. Gadgetcabaret, última actualización 5 de diciembre del 2007. Disponible en: http://gadgetcabaret.wordpress.com/2007/12/05/netbeans-60-instalacion-y-caracteristicas/.

18. Galván Jiménez Atención Familiar. Disponible en: http://www.facmed.unam.mx/deptos/familiar/atfm123/temasinteres.html.

19. Grandi y Asociados-Software para Consultorios Médicos. Disponiible en: http://www.grandiyasociados.com/software.asp?idSoftware=8&idIdioma=1.

20. Héctor Suárez González Manual Hibernate .Publicado el 21 de marzo de 2003. Disponible en: www.javahispano.org/contenidos/archivo/77/ManualHibernate.pdf.

21. Hispano Asociación Java Java Hispano . Java Hispano. Publicado en el 2002. Consultado el 14 de mayo del 2008. Disponible en: http://www.javahispano.org/contenidos/es/manual_hibernate/.

22. Intermedical.freeservers.com . Consultado el 20 de febrero del 2008.Disponible en: http://intermedical.freeservers.com/descripcion%20resumida%20intermedical%20soft.html.

23. itapizaco,Tutorial de Java, Swing . Consultado el 5 de mayo del 2008. Disponible en: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte14/cap14-1.html .

24. Jerssoft - GOF (Parte I) .Publicado el 1ro de julio del 2005. Disponible en: http://jerssoft.blogspot.com/2005/07/gofparte-i.html.

25. Laura Bermejo Sanz Enrique Gómez Monreal. Eclipse como IDE. Consultado el 03 de mayo del 2008. Disponible en: http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf.

26. NetBeans .Consultado el 5 de mayo del 2008. Disponible en: http://www.netbeans.org/index_es.html.

27. Pecos Daniel PostgreSQL vs. MySQL .Consultado el 8 de mayo del 2008. Disponible en: http://www.netpecos.org/docs/mysql_postgres/x15.html.

28. Pressman. Roger S**.** Ingeniería del Software. Un Enfoque Práctico. [Libro]. Publicado en el 2001. Quinta edición.

29. Probando software y números de versión. Autor Diego Lucio D'Onofrio, publicado en el 2002. Disponible en: http://www.elguille.info/Clipper/probando.htm.

30. Programa de Trabajo del Médico y la Enfermera de la Familia, el Policlínico y el hospital. Disponible en: http://aps.sld.cu/bvs/materiales/programa/progra_tarbajo/programatrabajo.pdf.

31. Rizzi Ing. Francisco Marcelo Itba, Complejidad Ciclomática. Consultado el 20 de mayo del 2008. Disponible en: http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZI-COMPLEJIDAD.pdf .

32. Rodolfo J. Stusser Beltranena Alfredo Rodríguez Díaz. La Informatización de la Atención Primaria de la Salud, 2006. Disponible en: http://bvs.sld.cu/revistas/mgi/vol22_4_06/mgi12406.htm.

33. Scientific Electronic Library Online. Disponible en: http://scielo.sld.cu/scielo.php

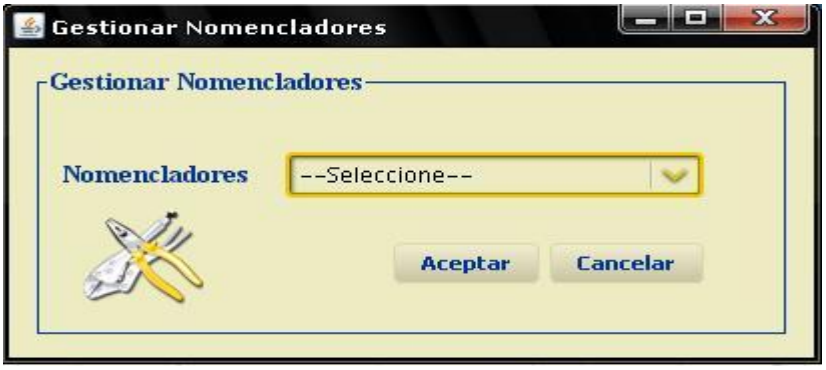
34. Seco José Antonio González devjoker,Tutorial de C#, última actualización 3 de octubre del 2006. Consultado el 10 de abril del 2008. Disponible en:

35. http://www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=125#a1.

ANEXOS

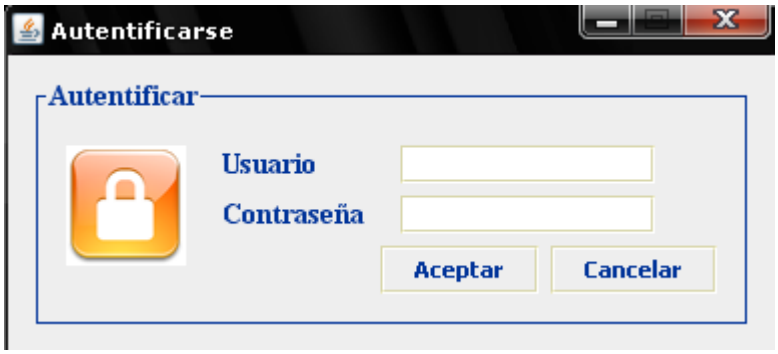
1. **Interfaz de ususario Principal.**

2. **Gestionar Nomencladores.**



3. Gestionar ususarios.

4. Autenticarse.



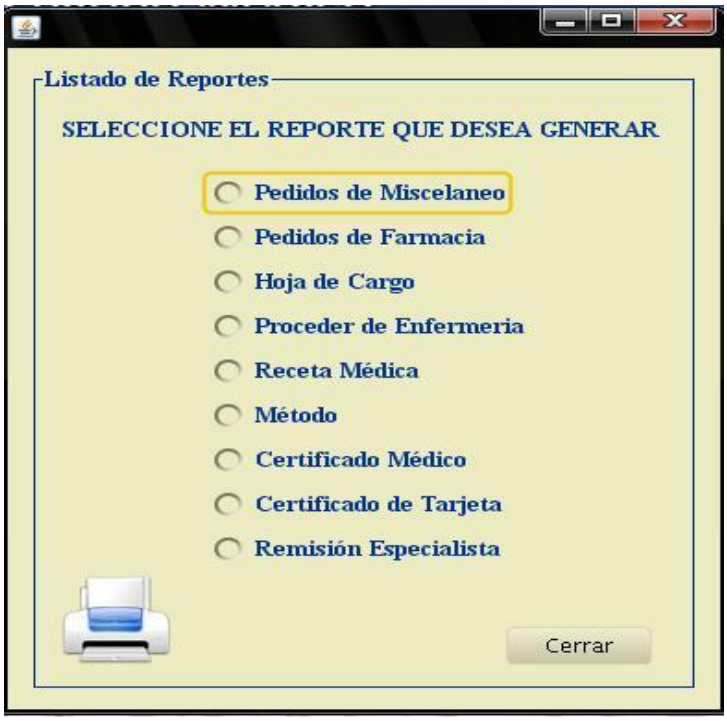
5.Crear Hoja de Cargo.

6.Buscar Paciente.

7.Historia Clínica.

8. Adicionar Pedidos.

9. Listado de Reportes.



GLOSARIO DE TÉRMINOS

Base de Datos (BD): es un conjunto exhaustivo no redundante de datos estructurados organizados, independientemente de su utilización y su implementación en máquina accesibles en tiempo real y compatibles con usuarios concurrentes, con necesidad de información diferente y no predicable en tiempo.

Clases: abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común.

Clases abstractas: Una clase que declara la existencia de métodos pero no la implementación de dichos métodos.

Disparadores (triggers): Un disparador define una acción que la base de datos debe llevar a cabo cuando se produce algún suceso relacionado con la misma.

General Public License o licencia pública general (GPL): es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

Health Level Seven (HL7): es una organización sin fines de lucro que desarrolla estándares para minimizar las incompatibilidades entre sistemas de información en salud, permitiendo la interacción y el intercambio productivo de datos entre aplicaciones heterogéneas, independientemente de su plataforma tecnológica o de su lenguaje de desarrollo.

Restricciones (Constraints): Constraints son restricciones en alguna columna en la definición de una tabla.

SQL: Lenguaje estandarizado de consultas utilizado para consultar bases de datos.

1