UNIVERSIDAD JOSÉ CARLOS MARIÁTEGUI

VICERRECTORADO DE INVESTIGACIÓN

FACULTAD DE INGENIERÍA Y

ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

TRABAJO DE SUFICIENCIA PROFESIONAL

JAVA Y BASE DE DATOS

PRESENTADO POR

BACHILLER SILVIA KARINA HUAMANÍ ANAHUA

ASESOR

ING. JULIAN MANUEL FLORES MANCHEGO

PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO DE SISTEMAS E INFORMÁTICA

MOQUEGUA-PERÚ

2019

CONTENIDO

CAPÍTULO III

DESARROLLO DEL TEMA

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

ÍNDICE DE TABLAS

Pág.

ÍNDICE DE FIGURAS

Pág.

RESUMEN

El presente trabajo tiene como finalidad desarrollar un software utilizando como lenguaje de programación Java, que gestione el sistema de atención de pacientes, y el control de ingresos manejando para esto una base de datos utilizando SQL Server 2008, con el fin de poder ofrecer atención oportuna y eficaz a sus pacientes. Actualmente la clínica no cuenta con un sistema eficiente que permita gestionar la totalidad de pacientes, historias clínicas, citas, pagos; entre sus principales deficiencias son: tarda en atender a los usuarios, demora en la ubicación de la historia clínica, no se tiene un control adecuado en la asignación de citas médicas. Para ello gracias a la identificación de los procesos mediante el RUP, para la asignación de tareas y responsabilidades dentro de la clínica se logró implementar de manera adecuada el software, permitiendo a nuestros usuarios finales satisfacer sus necesidades, permitiendo que el 90 % de los usuarios este satisfecho con la implementación del sistema lo cual permitirá a la gerencia de administración la mejor toma de decisiones.

Palabras clave: Java, sistema, SQL Server, software

ABSTRACT

The purpose of this work is to develop a software using the Java programming language, which manages the patient care system, and income control, using a SQL Server 2008 database, in order to be able to offer timely attention. And effective to their patients. Currently the clinic does not have an efficient system that allows managing all patients, medical records, appointments, payments; among its main deficiencies are: it takes time to attend to the users, delay in the location of the clinical history, there is no adequate control in the allocation of medical appointments. For this, thanks to the identification of the processes through the RUP, for the assignment of tasks and responsibilities within the clinic, the software was implemented in an appropriate way, allowing our final users to satisfy their needs, allowing 90 % of the users be satisfied with the implementation of the system which will allow Management the best decision making.

Keywords: Java, system, SQL Server, software

INTRODUCCIÓN

En la actualidad las tecnologías de información han adquirido un importante avance en su uso. Ello ha conllevado a que su uso sea frecuente en las diferentes áreas del quehacer de las personas y del desarrollo económico en las empresas, nuestra función como ingenieros es ofrecer nuevas soluciones de arquitectura de software y ofrecer una plataforma adecuada para la integración de los sistemas en la organización, esto conlleva a una toma de decisiones correcta.

Actualmente según las investigaciones de mercado el índice de sistematización en las entidades públicas es de 85 %, y es fundamental para brindar un servicio de calidad. Los sistemas son usados como estrategias para lograr la competitividad.

Para el presente trabajo de tesis me basare en el manejo de los procesos de control de pacientes e historias clínicas respectivamente en la clínica odontológica Ebenezer, para esto se utilizará como Lenguaje de Programación a Java y como Sistema de Gestión de Base de Datos SQL SERVER 2008 R2, el cual me brindará en el sistema una escalabilidad y performance adecuada.

CAPÍTULO II

GENERALIDADES

2.1 Planteamiento del problema

2.1.1 Definición del problema

Actualmente la Clínica “Ebenezer”, así como las clínicas odontológicas en general no cuentan con un sistema eficiente que les permita gestionar la totalidad de pacientes, historias clínicas, citas, pagos, etc. Teniendo los siguientes problemas diariamente:

- Tarda en atender a los usuarios, debido a la exhaustiva búsqueda de historias clínicas que debe realizarse previamente, por cada paciente o en su defecto a la pérdida de estas.
- Demora con la atención debido a que no se encuentra la historia clínica o pérdida de esta.
- El archivamiento de las historias clínicas requiere excesivo espacio y tratamiento especial como folders o cajones especiales para su conservación.
- No se puede tener eficientemente el control de historias clínicas.

- Es imposible tener un riguroso control en la asignación de turnos otorgados a los pacientes para citas programadas (mala coordinación de los horarios de atención).
- No se lleva el control de pago de los servicios dados a los pacientes.

Por todo lo anteriormente descrito es necesario la implementación de un software que permitirá tener los datos como historias clínicas de cada paciente de manera digitalizada, ahorrando el espacio, evitando que se deterioren por el paso del tiempo, así como la emisión reporte diariamente.

2.1.2 Formulación del problema

¿Cuáles son los factores que causan la demora en manejo de los procesos de control de pacientes e historias clínicas respectivamente en la clínica odontológica Ebenezer?

2.2 Objetivos

2.2.1 Objetivo general

Desarrollar un software que gestione el sistema de atención de pacientes, y el control de ingresos manejando para esto una base de datos que contendrá el registro de todos los pacientes y del personal que labora en la clínica para poder así ofrecer atención oportuna y eficaz a sus pacientes.

2.2.2 Objetivos específicos

Analizar los principales procesos y actividades del sistema manual de atención a los pacientes de la clínica.

Diseñar el modelo físico y lógico de la base de datos.

Diseñar el software utilizando como lenguaje de programación Java

Utilizar como sistema de gestión de base de datos SQL SERVER 2008 R2 para desarrollar el modelo físico de la base de datos, procedimientos almacenados y administración de usuarios.

2.3 Justificación

Mediante el diseño, desarrollo e implementación del sistema se agilizarán los distintos procesos con los que cuenta la Clínica aminorándose el tiempo de atención del paciente, apresurando el proceso de búsqueda de las historias clínicas, llevando el control apropiado de las citas médicas; con el propósito de brindar comodidad y bienestar a los pacientes, cooperando de esta manera en el fortalecimiento de su nivel de competitividad y servicio de la clínica.

Caso contrario, de no realizarse la implementación del sistema que ofrezca las prestaciones tecnológicas necesarias para mejorar los problemas que se han detectado, la clínica en general no podrá competir frente a otras clínicas, perdiendo así recursos económicos y recursos humanos, los mismos que son el eje del desarrollo de la clínica.

2.4 Alcances del estudio

El presente trabajo de investigación: “Elaboración de un sistema de control de citas médicas de la clínica dental Ebenezer”, desarrollado en Java y SQL Server se definen los requerimientos (funcionales, no funcionales) se ocupa de clasificar y priorizar cada uno de los requerimientos dados por el usuario. A su vez la diagramación de los planes de la estructura del sistema mediante el leguaje de

modelamiento unificado (UML). Los cuales son diagrama de caso de uso, diagrama de paquetes, diagrama de secuencia y diagrama de arquitectura.

Asimismo, el documento realizado tiene un alcance solo en la clínica dental. Los cuales están considerados los siguientes procesos.

- Consulta, asignación de citas
- Atención, control de citas
- Control de pagos

2.5 Estrategia metodológica

2.5.1. Revisión bibliográfica

En este punto realizaremos las siguientes actividades:

- Visitas a bibliotecas privadas como las de las universidades estatales y particulares como la UPT, JBG, TELESUP, etc.
- Búsqueda en internet en páginas web referidas a sistemas de trámite documentario, páginas sobre metodologías de control de documentos y otros, revistas online, tesis, etc.
- Revisión de la documentación interna de la empresa con respecto al manejo del control de documentación de la OTI.
- Investigación de cursos, talleres sobre control de documentación.

2.5.2. Diseño de la investigación

Este proyecto siguió esta metodología para diseñar una herramienta adecuada para un control y gestión de la clínica.

También, se empleó como metodología el RUP (proceso unificado de rational) y el lenguaje unificado de modelado UML; el primero para el desarrollo

del software, considerando sus atributos de flexibilidad y adaptabilidad al contexto y necesidades del caso estudiado; y el segundo para el análisis, implementación y documentación de sistemas orientados a objetos.

2.5.3. Desarrollo de la propuesta

Para desarrollar la propuesta de solución, se realizarán las siguientes actividades:

- Se estudia otros casos similares al registro y seguimiento de documentación, para así facilitar el manejo del trabajo.
- Se efectúa la revisión de tecnologías de información actual referida para el análisis del sistema, sobre todo aquellas herramientas comerciales que permitan enriquecer la propuesta tecnológica que se implementara el sistema.
- Se especifica una propuesta de solución donde se define y justifica el problema a resolver, y se plantea la solución.

2.5.4. Validación de la propuesta

Para validar la propuesta realizamos lo siguiente:

- Se llevan a cabo reuniones con los usuarios con la finalidad de validar la solución propuesta, ya que se debe satisfacer las necesidades directamente de los usuarios de la aplicación de escritorio, así como verificar que propuesta brinde los resultados necesarios a la empresa.
- Se realizan consultas al asesor del trabajo, con la finalidad de establecer una evaluación objetiva sobre la propuesta realizada.
- Establecer conclusiones y recomendaciones una vez validada la propuesta se deberá escribir todas las conclusiones y recomendaciones para la Empresa.

CAPÍTULO III

DESARROLLO DEL TEMA

3.1. Lenguaje de programación: Java

Java es un lenguaje de programación, característico por su reducido tamaño, portabilidad y de plataforma libre a nivel de código fuente y binario; siendo posible su interpretación en cualquier plataforma de programación, sin necesidad de reescribir su código. De igual forma, los bytecodes que constituyen los archivos binarios compilados, equivalentes al código máquina general, pueden ejecutarse en cualquier plataforma y no requieren de recopilación alguna. Para la ejecución de programas en Java, es necesario efectuar previamente dos operaciones; primero, la compilación para obtener los bytecodes mediante el compilador javac.exe; y finalmente, para su interpretación en la plataforma mediante el intérprete Java. Además, se caracteriza por ser multitarea, pues permite la ejecución simultánea de varios programas en el navegador; permitiendo la creación de aplicaciones que realicen tareas simultánea. Sin embargo, sólo determinados sistemas operativos soportan esta función, como UNÍX o Windows NT (Morgan, 1999).

También se dice que es un lenguaje dinámico, debido a su capacidad de interacción con el usuario; pudiendo desarrollarse interfaces gráficas potentes de

usuario para Applets y aplicaciones de consola. A principios de la década de 90, James Gosling y sus compañeros, desarrollaron el lenguaje de programación bajo la orientación por objetos. Generalmente, para realizar la compilación se emplea el compilador JIT, el mismo que es ejecutado desde la máquina virtual Java. Su lenguaje es similar a C y C++, pero su modelo orientado a objetos resulta ser más sencillo, y ha sido influenciado por Smalltalk y Eiffel. Java Specification Request, bajo las siglas JSR 14; sugiere la incrustación de tipos y métodos genéricos en el lenguaje de programación Java. Sin embargo, ya desde sus inicios, los desarrolladores trataron de añadir tipos genéricos al lenguaje; siendo estos últimos empleados durante años en otros lenguaje y ahora forman parte de Java 1.5. El paquete JDK, incluye el entorno de desarrollo de Java de Sun; el cual, resulta útil para el desarrollo de programas Java y propia el entorno indispensable para su ejecución. Además, comprende dos tipos; binario precompilado y paquete fuente. Al realizar la compilación del código fuente JDK, es necesario descargar la versión binaria para la elaboración del JDK; siendo necesario descargar cuatro ficheros para poder completar la construcción de fuentes: jdk-1_5_0_02-linux-i586.bin; jdk-1_5_0-src-jrl.zip; jdk-1_5_0-bin-jrl.zip; y jdk-1_5_0-mozilla_headers-unix.zip, jdk-6-rc-windows-i586 (Morgan, 1999).

3.1.1. Características de Java

Según Morgan (1999), las siguientes son las características de Java.

3.1.1.1 Simple.

Cuenta con las características de un lenguaje potente, a excepción de:

- Aritmética de punteros.
- Liberación de memoria

- Reducción del 50 % de errores más comunes en la programación al eliminar las otras dos características mencionadas anteriormente.

3.1.1.2 Orientado a objetos.

Exige ser realizada bajo la orientación a objetos pura, no siendo permita la programación procedural.

- De arquitectura neutral: Compila su código a un archivo objeto, resultando su formato autónomo a la arquitectura de la máquina en la que se ejecute.
- Robusto: Verifica de forma continua la existencia de problemas, durante su compilación y ejecución.
- Distribuido: Cuenta con extensas capacidades de interconexión TCP/IP y librerías de rutinas, que permiten el acceso e interacción con protocolos como http y ftp.

3.1.1.3 Arquitectura neutral.

Para que Java forme parte integral de la red, previamente se debe compilar su código a un archivo objeto; y por ser de formato autónomo a la arquitectura de la máquina en que se ejecuta, puede ser ejecutado en cualquier máquina que cuente con el sistema run-time, sin importar en donde fue generado. Con Sistema, nos referíamos a la Máquina Virtual Java (JVM) y sus librerías fundamentales, que permiten el acceso directo al hardware de la máquina. Asimismo, cabe precisar que los APIs de Java que contacten de forma directa con el hardware, son dependientes de la máquina, por ejemplo:

- Java 2D: Gráficos 2D y manipulación de imágenes.
- Java media framework: Elementos críticos en el tiempo (audio, video).
- Java animation: Animación de objetos en 2D.

- Java telephony: Integración con telefonía.
- Java share: Interacción entre aplicaciones multiusuario.
- Java 3D: Gráficos 3D y su manipulación.

3.1.1.4 Seguro.

La seguridad comprende dos facetas. Respecto al lenguaje, las características como punteros o casting implícito generados por los compiladores de C y C++ son eliminados para impedir el acceso ilegal a la memoria. Además, cuando empleamos Java para la creación de un navegador, se combinan las características mencionadas con la protección común aplicable al navegador específico.

Cuando los byte-codes no generan mensaje alguno de error durante la verificación, podemos afirmar que:

- El código no produce desbordamiento de operandos en la pila.
- El tipo de los parámetros de todos los códigos de operación son conocidos correctos.
- No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros.
- El acceso a los campos de un objeto se sabe que es legal: public, private, protected.
- No existe intento alguno de violación de reglas de acceso y seguridad establecidas.

3.1.1.5 Portable.

Cuenta con otros estándares de portabilidad a fin de facilitar su desarrollo.

3.1.1.6 Interpretado.

El sistema run-time puede ejecutar de forma directa el código objeto. Generalmente, la ejecución de programa requiere menos recursos que su compilación; es por ello que los desarrolladores de Java, dedicaron mayor tiempo en su desarrollo y menos en la espera del ordenador. Java es considerado 10 - 30 veces más lento que C, además, a diferencia de otros lenguajes, no abarca proyectos de gran envergadura.

3.1.1.7 Multithreaded.

Al ser multithreaded (multihilvanado, en mala traducción), posibilita la realización de diversas actividades de forma simultánea en un mismo programa. Los threads, denominados como procesos ligeros, son pequeños procesos o piezas independientes de un gran proceso, además su uso es más sencillo y son más robustos en comparación al lenguaje en C o C++. La ventaja de ser miltithreaded se ve reflejada en mejoras sustanciales en su rendimiento interactivo y su comportamiento en tiempo real.

3.1.1.8 Dinámico.

Maximiza sus beneficios al tope, respecto a la tecnología orientada a objetos. No conecta los módulos de la aplicación hasta que sea ejecutada. Las librerías nuevas o actualizadas no paralizan las aplicaciones actuales.

3.1.2. Plataformas

Según Morgan (1999), son conceptualizadas como, el ambiente de hardware o software en el que los programas son ejecutados. Mayormente, pueden ser explicadas como la mezcla del hardware con el sistema operativo. Dentro de las más conocidas, podemos mencionar a Windows, Solaris, Linux y MacOS.

Java, constituye una plataforma virtual que se caracteriza por proveer:

- El lenguaje Java: es lenguaje de programación.
- El Java Virtual Machine: Máquina virtual que cuenta con su propio set de instrucciones.
- La Java Platform API: Interfaz que permite la programación de aplicaciones.

3.1.3. Otras plataformas Java

Según Morgan (1999), existen otras plataformas Java:

3.1.3.1. JavaCard.

Destinado a posibilitar el desarrollo de aplicaciones que vayan a ser empleadas en tarjetas inteligentes.

3.1.3.2. EmbeddedJava.

Posibilita el desarrollo de aplicaciones que guardan relación con el acceso a red, específicamente la entrega de servicios bajo demanda. Para mayor ilustración, mencionamos algunos ejemplos a continuación: máquinas de venta ambulatoria, dispensador de combustible, siendo el servicio efectuado mediante la red; cabe señalar que este tipo de máquinas, requieren apenas 500 Kbytes como máximo.

3.1.3.3. JavaPhone.

Posibilita el desarrollo de aplicaciones para teléfonos futuristas. Podemos citar como ejemplo a las bibliotecas de clases.

Respecto a donde serán ejecutadas, podemos clasificarlas en dos, según el tipo de teléfono:

- Teléfonos inteligentes: Permiten de forma conjunta la comunicación por voz, radio, fax, correo electrónico, aunado al acceso a Internet, planifica tareas al estilo de PDAs y otras más.

- Teléfonos con pantalla para Internet: Cuentan con pantallas de vídeo y alternativamente teclados, los cuales posibilitan la comunicación personal vía Internet.
- Java TV: Se refiere a la televisión de forma digital e interactiva, que nos proporciona una plataforma para desarrollar programas que permitan controlar la televisión y los "set-top boxes" creados para el entretenimiento digital, resultando ser independiente a la tecnología subyacente empleada para la emisión.

3.1.4. Lenguaje de ayuda: Netbeans 8.0

Según Morgan (1999), NetBeans es un entorno para desarrollar en multilenguaje, siendo comúnmente empleado por los nuevos programadores y los profesionales.

Constituye un código abierto, es decir, de uso libre o no pago, su creación fue realizada por Java, SUN, su descarga e instalación son sencillas conjuntamente con el J2SE que es un paquete de desarrollo. Esta plataforma es empleada para desarrollar aplicaciones de escritorio mediante Java y un entorno de desarrollo integrado, conocido por las siglas IDE, el cual fue desarrollado mediante la Plataforma NetBeans. Se le considera como la plataforma que posibilita el desarrollo de aplicaciones desde un conjunto de componentes de software denominados módulos. Podemos conceptualizar a módulo como, el un archivo Java que comprende las clases de Java escritas para la interacción con las APIs de NetBeans y un archivo especial que hace posible su identificación como módulo. Las aplicaciones que fueron desarrollar desde módulos podrán ser extendidas, es decir, que se les puede agregar nuevos módulos. Ya que, los módulos se pueden desarrollar de forma independiente, las aplicaciones basadas

en NetBeans podrán ser extendidas con facilidad por otros desarrolladores de software. Actualmente, cuenta con una extensa base de usuario. Con las versiones 6.01 y 6.5 se otorgó soporte a frameworks comerciales como struts e hibernate.

Su plataforma, es una base modular y extensible empleada como estructura de integración con la finalidad de desarrollar grandes aplicaciones de escritorio. Las empresas que son especialistas en desarrollar softwares y cuentan con calidad de socias e independientes, suelen otorgan extensiones adicionales que pueden incorporarse con facilidad a la plataforma y ser utilizadas para el desarrollo de herramientas y soluciones particulares para sí mismas. Además, proporciona servicios generales para las aplicaciones de escritorio, que le permite al desarrollar poder centrarse específicamente en la lógica de la aplicación IDE NetBeans.

3.1.5. Características de netbeans 8.0

Según Morgan (1999), las características más importantes propias de la plataforma de netbeans, son:

- Gestión de interfaces del usuario (por ejemplos: los menús y las barras de herramientas).
- Gestión de configuración de usuario.
- Gestión de almacenamiento (permite guardar y cargar todo tipo de datos).
- Gestión de ventanas.
- Framework basado en asistentes.

3.2. Lenguaje de modelado unificado

El Lenguaje de Modelado Unificado, conocido por su abreviatura en siglas UML, que responden a la denominación en inglés de Unified Modeling Language; es definido como el lenguaje de modelado de sistemas de software conocido y

utilizado en la actualidad; se encuentra respaldado por el Object Management Group, que responde a las siglas OMG. Además, permite la representación, explicación, fabricación y documentación de sistemas. UML proporcionar un estándar para la descripción en “plano” del sistema, conocido también como modelo, incorporando criterios conceptuales como los procesos de negocio y funciones propias del sistema, asimismo, aquellos aspectos concretos como las expresiones de lenguaje de programación, esquemas de bases de datos y componentes reutilizables (Fowler y Scott, 1999).

Cabe resaltar, la importancia de UML como "Lenguaje de Modelado", pues resulta fundamental para la explicación o descripción de métodos o procesos. Es utilizado en la definición de sistema, especificación de artefactos en el sistema y así como para la documentación y construcción. En resumen, constituye el lenguaje en el que se describe el modelo. Resulta aplicable para desarrollar softwares, debido a la diversidad de formas de dar soporte con las que cuenta respecto a la metodología de esta, como por el Proceso Unificado Racional (RUP); sin embargo, no precisa la metodología o proceso se debe emplear (Fowler y Scott, 1999).

De igual forma, es incomparable a la programación estructura, ya que el UML no es programación, sino el diagrama de la realidad de una utilización en un determinado requerimiento, existiendo gran diversidad de estos que reflejan los diversos aspectos de la entidad representada; mientras que la otra es considera como la forma de programa bajo la orientación a objetos, pese a ello resulta ser un complemento extraordinario para el UML (Fowler y Scott, 1999).

3.2.1. Diagramas

Según Fowler y Scott (1999), la versión 2.0 de UML, considera 13 tipos de diagramas de estructura, los cuales se caracterizan por enfatizar los elementos necesarios que deben ser considerados según el sistema modelado:

- De clases.
- De componentes.
- De objetos.
- De estructura compuesta.
- De despliegue.
- De paquetes.

Mientras que, los diagramas de comportamiento realzan la importancia de los sucesos según el sistema modelado, pudiendo ser:

- De actividades.
- De casos de uso.
- De estados.

Asimismo, en la figura 1 se muestra los Diagramas de Interacción forman parte de los diagramas de comportamiento, como un tipo dentro de estos, que se caracteriza por hacer énfasis en el flujo de control y los datos entre elementos del sistema modelado.

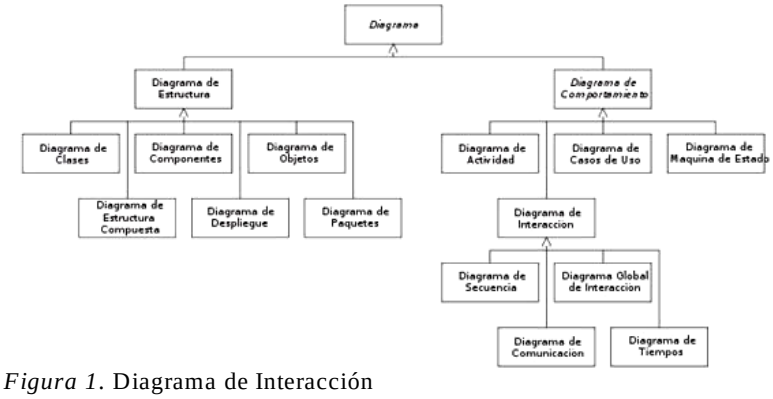

Figura 1. Diagrama de Interacción
Fuente: Fowler y Scott, 1999.

3.2.2. Un estudio a fondo del UML

Según Fowler y Scott (1999), en los sistemas de reversa de aerolíneas simples, suele ser empleado para exhibir o revelar los diagramas y técnicas de modelado con UML. Donde, se cubrirán básicamente los puntos referidos a:

- Organización del sistema por paquetes.
- Modelado con Casos de Uso, y empleándolos para determinar los requisitos del sistema.
- Modelado con Diagramas de Secuencia y Colaboración.
- Análisis y diseño mediante el Diagrama de Clase, y extensión de UML por la técnica de tarjetas CRC.
- Modelado del comportamiento mediante Diagramas de Actividad y de Estado.
- Modelado de componentes del software, respecto a la distribución e implementación.
- Extensión del UML mediante el diseño de Bases de Datos relacionales.

Dentro de las principales tareas para la modelación de un sistema de software de grandes dimensiones, se encuentra la división en pequeñas áreas, siendo esta última denominada como subsistemas, dominios o categorías. Prevaleciendo la idea de división del sistema por áreas, agrupadas según la similitud de competencias.

Asimismo, con el UML se introdujo la concepción del paquete como ítem universal para la agrupación de elementos, posibilitando a los modeladores para la subdivisión y categorización de sistemas. Pudiendo ser empleados en distintos niveles, en los niveles más altos son empleados para la subdivisión del sistema en

dominios y en el nivel más bajo para la agrupación de casos de uso individual, clase o componente.

3.2.3. Modelado de caso de uso

Según Fowler y Scott (1999), es la técnica con mayor efectividad y a su vez la que posee mayor sencillez para la modelación de requisitos de sistema desde la óptica de los usuarios. Este tipo de modelado se utiliza para la modelación como sistema o negocio, el cual viene funcionando en la actualidad; o como el usuario desea que funcione. No es similar ni cercano a la orientación a objetos, es más como una forma de modelación por procesos. Los casos uso son generalmente el origen del análisis orientado a objetos mediante UML. Este tipo de modelado, comprende actores y casos de uso. Los primeros representan al usuario y demás sistemas que interactúen con el sistema. Se representan gráficamente como "muñecos" de palo. En la actualidad, simbolizan los tipos de usuarios, más no instancias de estos; mientras, que, los casos simbolizan el comportamiento del sistema, aquellos escenarios que han de ser atravesados como producto del estímulo a partir de un actor, gráficamente se representan con elipses.

Debiendo ser cada uno documentado por la descripción del escenario, pudiendo ser escrita en forma de texto o en otro formato de paso a paso. Además, pueden ser definidos en base a otras propiedades, como son las condiciones previas y posteriores del escenario, condiciones que se encuentran con anterioridad al inicio del escenario, y las que se encuentran después que el escenario es completado. Es así que, los Diagramas de Actividad proporcionan una herramienta gráfica para la modelación del proceso de Caso de Uso.

3.2.4. Descubrimiento de requisitos

Según Fowler y Scott (1999), la finalidad principal del diseño de software es la satisfacción de requisitos de los usuarios para el sistema. Estos, pueden referirse al software, productos o pruebas. A fin de poder reflejar y corroborar que los requisitos exigidos por el usuario sean considerados en el diseño y se encuentre conforme a estos.

3.2.5. Organización de diagramas de casos de uso

Según Fowler y Scott (1999), en el análisis de negocio (business) del sistema, se puede elaborar el modelo de caso de uso para este, y elaborar paquetes que representen los distintos dominios de negocio (business) del sistema.

Es posible descomponer cada uno de los paquetes con su Diagrama de Caso de Uso, el cual deberá contener los Casos de Uso de un dominio, con intervenciones de actor.

3.2.6. Un caso de uso por cada escenario

Según Fowler y Scott (1999), la finalidad es la elaboración de un Diagrama de Caso de Uso por cada uno de los tipos de escenario diversos del sistema; donde, cada uno deberá mostrar una secuencia diversa de intervenciones de actores y sistema, omitiendo la condición “or”.

3.2.7. Modelar secuencias alternas a través de la relación "Extiende" (Extends)

Según Fowler y Scott (1999), frecuentemente, la modelación de cada Caso de Uso se realiza mediante una secuencia normal de acciones. Donde, el usuario considerará condicionantes "que si" para cada uno de los pasos, y se desarrollará en base a secuencias alternas de eventos. Estas últimas, son modelados por casos

de uso independientes, los que se relacionan con el caso de uso original por la relación "Extiende" (extends). Estas relaciones podrán ser planificadas como caso de uso que resulta ser la equivalencia de herencia, en el que el caso de uso extendido, hereda y modifica el comportamiento del caso de uso inicial.

3.2.8. Eliminación del modelado redundante mediante la relación "Usa" (Uses)

Según Fowler y Scott (1999), la eliminación del modelado reiterativo de buena parte del comportamiento que figure en múltiples casos de uso, la fracción correspondiente al comportamiento podrá ser modelada en un caso de uso separado que se relacionará con los demás casos de uso con la relación "Usa" (uses); esta relación puede ser ideada como el caso de uso similar a “of aggregation”.

3.2.9. Casos de uso probando el sistema frente a los requisitos

Según Fowler y Scott (1999), el Caso de Uso suele ser empleado también para la elaboración de scripts de prueba, que suelen ser usado además para la verificar que la aplicación cumple con los requisitos del negocio y del sistema. En el supuesto de haber llegado al nivel más bajo, es posible la creación de un Diagrama de Secuencia para el Caso de Uso; correspondiendo los de secuencia y colaboración para la modelación de la implementación del escenario.

3.2.10. Diagramas de secuencia

Según Fowler y Scott (1999), es un tipo de diagrama y se considerada como el más real para la modelación de interactuar entre objetos del sistema; suele ser diseñado para cada caso de uso. En cambio, el diagrama de caso de uso permite la modelación de una vista ’business’ del escenario; y el diagrama de secuencia

comprende detalles de implementar del escenario, incluye objetos y clases que son empleados para la implementación del escenario, y los mensajes a ser transmitidos por los objetos. Usualmente, se examinan la descripción de un caso de uso para la determinación de los objetos útiles para implementar el escenario. De contarse con la descripción modelada de cada caso de uso mediante secuencia de los distintos pasos, se puede "caminar sobe" aquellos pasos para el descubrimiento de los objetos fundamentales para poder continuar los pasos.

Estos diagramas, muestran a los objetos intervinientes en el escenario mediante líneas verticales discontinuas, y a los mensajes transmitidos entre objetos en forma de vectores horizontales. Los mensajes se representan de forma cronológica, consignado primero la parte superior del diagrama hasta la parte inferior; su repartición horizontal es impuesta. Mientras se efectúe el análisis inicial, el modelador usualmente consignará la denominación de "business" como mensaje en la línea respectiva. Luego, mientras se realice el diseño, dicha denominación será sustituída por el nombre del método que esté siendo llamado por un objeto en el otro. Aquel método invocado o llamado, ha de pertenecer a la conceptualización de la case instanciada por el objeto en la recepción final del mensaje.

3.2.11. Diagramas de colaboración

Según Fowler y Scott (1999), estos diagramas son una opción, respecto al diagrama de secuencia para la modelación de intervenciones entre los objetos del sistema. Es así que, el diagrama de secuencia se enfoca en la secuencia cronológica del escenario que se encuentra en proceso de modelación, mientras que el de colaboración se focaliza en el estudio de los efectos de un determinado

objeto durante el escenario. Los objetos intervienen mediante enlaces, y cada uno de estos es la representación de las instancias existentes entre las clases implicadas. Los enlaces muestran que mensajes han sido remitidos a los objetos, su tipo (sincrónico, asincrónico, simple, blanking, y "time-out"), y sí el objeto es visible en referencia a los demás.

3.2.12. Análisis y diseño con el diagrama de clase

Según Fowler y Scott (1999), es el diagrama central en el diseño y análisis de un sistema. Su estructura de clases es determinada, sus relaciones mediante clases y estructuras de herencia. En el análisis del sistema, el diagrama es elaborado bajo el enfoque de alcanzar la ideal solución. Mientras se efectúa el diseño, se emplea el mismo diagrama, y se efectúan las modificaciones necesarias para el cumplimiento de las especificaciones correspondientes a las implementaciones.

3.2.13. Diseño de componentes

Según Fowler y Scott (1999), el componente es considerado como el conjunto de objetos o componentes más pequeños que intervienen entre sí y se combinan a fin de otorgar un servicio. Son semejantes a una caja negra, en la que los servicios correspondientes al componente se detallan por interfaces o interface, sin otorgar conocimiento respecto a su diseño o implementación interna. La realización en base a componentes, constituye el proceso de ensamblaje de la mezcla adecuada de componentes en la configuración correcta para realizar las funciones deseadas en un sistema. Los componentes son representados en el diagrama de clases de UML precisando su interfaz de paquete o clase.

Existen un par de notaciones que muestran la interfaz – una es revelar la interfaz como "regular class symbol" bajo el estereotipo "interfaz", con el listado

de operaciones que son soportadas por esta, precisando en el "operation department" (departamento de operación). "The 23ltérnate, shortcut notation" es representar a la interfaz mediante un pequeño círculo conjuntamente con la clase mediante línea sólida, señalando la denominación de la interfaz dentro del círculo.

3.2.14. Modelación del comportamiento de la clases

Según Fowler y Scott (1999), existen diferentes tipos de diagramas:

3.2.14.1. Diagramas de estado.

Este diagrama es empleado para el diseño del comportamiento dinámico de un objeto en específico o una clase de estos. Además, su diseño responde a todas las clases que sean consideradas con comportamiento dinámico. En este, se modelan las secuencias de estado que un objeto de la clase atraviesa mientras su vida es determinada por el efecto producido por los estímulos recibidos, conjuntamente a las acciones y respuestas.

3.2.14.2. Diagramas de actividad.

Es el diagrama que representa el flujo del proceso multi-propósito, se emplea para la modelación del comportamiento del sistema. Además, podrán ser empleados para la modelación de Caso de Uso, método complicado o clase. Son similares a los diagramas de flujo; con la diferencia que estos muestran el procesamiento paralelo (parallel processing). Resultando fundamental en el supuesto de uso para modelación de procesos 'bussiness', de los cuales unos podrán actuar de forma paralela y para la modelación de múltiples hilos en programas concurrentes.

3.2.14.3. Modelación de componentes de software.

Los diagramas de componente se suelen emplear para la modelación de la estructura de software, inclusive de sus dependencias entre componentes, sus

componentes de código binario, y los ejecutables. Mientras, los Diagramas de Componentes son empleados para modelar componentes del sistema, algunas ocasiones mediante agrupación en paquetes y sus dependencias existentes por componentes y paquetes de estos últimos.

3.2.14.4. Modelación de distribución e implementación.

El Diagrama de Implementación es usado para la modelación de configuración de aquellos elementos que son procesados durante la ejecución (run-time processing elements) y aquellos procesos, componentes y objetos del software que existen en estos. El diagrama 'deployment', inicia con la modelación de los nodos físicos y de las asociaciones de comunicación que hay entre estos. Por cada nodo, se podrá señalar las diversas instancias de Componentes que existen o son ejecutados por el nodo. Además, pueden modelarse aquellos objetos contenidos por componente. Estos diagramas, son empleados para la modelación de componentes que hay como entidades durante la ejecución; no han de ser empleados para la modelación de componentes durante la compilación o durante el enlazamiento. Pudiendo, además efectuar la modelación de componentes que van de nodo en nodo o aquellos objetos que van de componente en componente empleando la relación de dependencia bajo el estereotipo 'becomes' (se transforma)

3.2.15. Arquitectura del sistema (Cliente – Servidor)

Según Fowler y Scott (1999), es el modelo de aplicación distributiva, porque sus tareas son repartidas por proveedores de servicios o recursos, denominados servidores, y sus demandantes son denominados clientes. El cliente efectúa las peticiones a otro programa, mientras que el servidor es el encargado de emitir la respuesta. Pudiendo ser aplicada en otros programas que sean ejecutados en una

sola computadora, pero resulta ser más ventajosa en los sistemas operativos multiusuario distribuidos mediante red de computadoras.

3.2.16. Programación por capas

Según Fowler y Scott (1999), se le define como el estilo de programación, característico porque su fin esencial es la división lógica de los negocios y el diseño; el caso más común es la separación en capa de datos y capa de presentación para el usuario. Su mayor bondad es su realización por niveles y, en caso de modificación alguna, únicamente implica al nivele determinado sin necesidad de revisar el código combinado. El modelo de interconexión de sistemas abiertos es una clara muestra de aplicación de este método. Asimismo, posibilita la distribución del trabajo de creación de la aplicación en niveles; consecuentemente, cada uno de los grupos de trabajo será completamente abstraído de los demás niveles, de modo que sólo bastará con tener conocimiento de la API existente entre los diversos niveles.

Para diseñar los sistemas informáticos, actualmente se emplea la programación por capas o arquitecturas multinivel. En estas últimas, se les es confiada una misión simple, permitiendo que el diseño sea escalable (es decir, posibilita su amplificación de forma sencilla bajo el supuesto que las necesidades sean incrementadas). El diseño en tres capas o niveles es el que se emplea con mayor frecuencia en la actualidad.

3.2.17. Capas y niveles

Según Fowler y Scott (1999), se le conoce como capa de presentación de usuario a la que es visualizada por el usuario, es decir, aquella que muestra el sistema a los usuarios, transmite información y capta la información del usuario en un mínimo

de proceso, es decir que efectúa un filtro de forma previa para corroborar que no existen errores de formato. De igual forma, se le conoce como interfaz gráfica y principalmente se debe caracterizar por ser "amigable", quiere decir de usos sencillo, fácil y entendible para los usuarios. La presente capa solamente guarda comunicación con la capa de negocio. La capa de negocio o lógica del negocio, es definida como aquella en la se encuentran los programas en ejecución, recepciona las solicitudes de los usuarios y manda la respuesta posterior al proceso. Además, es en esta que se determinan las normas que habrán de ser cumplidas; guarda comunicación con la capa de presentación, a fin de recepcionar las peticiones y mostrar los resultados; y con la capa de datos respecto a los requerimientos dirigidos a gestor de base de datos para el almacenamiento o recuperación de datos a partir de este, dentro de este se consideran a los programas de aplicación.

La capa de datos, es conceptualizada como la contenedora de datos y la encargada de su acceso. Se encuentra conformada por uno o más gestores de bases de datos que efectúan el almacén de datos, recepcionan las peticiones de almacén y recuperamiento de información a partir de capa de negocio. La totalidad de capas pueden encontrarse presentes en un solo ordenador, aunque lo más común es la existencia de gran cantidad de ordenadores en los que se encuentre la capa de presentación, se refiere a los clientes de la arquitectura. Las capas de negocio y datos pueden encontrarse en un mismo ordenador, y de ser necesario debido al incremento de necesidades, es recomendable la separación en dos o más ordenadores. De modo que, en caso de incrementarse la complejidad de la base de datos, se podrá dividir en diversos ordenadores, los mismos que recepcionarán las peticiones del ordenador en el que se encuentre la capa de negocio.

De ocurrir lo contrario, quiere decir que la complejidad se encontrase presente en la capa de negocio, de modo que fuese necesaria la división, esta capa podría encontrarse en uno o más ordenadores que se encargaría de efectuar los requerimientos desde una sola base de datos. En caso de sistema demasiado complejo, se tiene diversidad de ordenadores para que corra la capa de negocio, y otros para la base de datos.

3.3. Sistema de gestión de base de datos SQL Server

Según Vargas (2013), el sistema de Microsoft SQL Server, es utilizado para gestionar las bases de datos producidas en Microsoft, en base al diseño relacional. Su lenguaje de consultas T-SQL y ANSI SQL. Microsoft SQL Server en la opción de Microsoft para los demás sistemas gestores que son consideradores como potentes, entre los que se encuentran Oracle, PostgreSQL o MySQL.

- Características de Microsoft SQL Server
- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Tolera procedimientos almacenados.
- Comprende entorno gráfico de administración potente, que posibilita el empleo de comandos DDL y DML gráficamente.
- Posibilita el trabajo de forma cliente-servidor, ya que la información y los datos son almacenados en el servidor y los clientes o terminales de la red únicamente podrán acceder a la información.

Asimismo, posibilita la administración de información a partir de diversos servidores de datos. SQLCMD es la línea de comando que se emplea para el uso de SQL. En la realización de aplicaciones de mayor complejidad, es decir,

aquellas de tres o más capas. Microsoft SQL Server comprende diversas interfaces de acceso para diversas plataformas de desarrollo, como NET. Sin embargo, el servidor únicamente se encuentra disponible para los Sistemas Operativos.

3.3.1. Programación T- SQL

Según Vargas (2013), se le define como la primordial forma de intervención con el Servidor. Permitiendo la realización de aquellas operaciones clase en SQL Server, incluyendo crear y modificar los esquemas de base de datos, lo referido a la introducción y edición de datos en la base de datos; asimismo, la administración del servidor como tal. Debiendo ser ejecutada a través del envío de sentencias de T-SQL y aquellas declaraciones que son ejecutadas por el servidor y los errores o resultados puedan regresas a la aplicación cliente.

3.4. Rational unified process (RUP)

3.4.1. Historia

Según Jacobson, Booch y Rumbaugh (2000), en la figura 2 se muestra la línea temporal de la historia del RUP. Donde, Ivar Jacobson con la Metodología Ericsson es el suceso más importante y se encuentran en periodo 1967, son un acercamiento del desarrollo basado en componente, el cual introduce la concepción de Caso de Uso. Durante los periodos comprendidos entre 1987 a 1995, Jacobson constituyó la compañía Objectory AB y lanzó el proceso de desarrollo de Objectory (abreviatura de Object Factory).

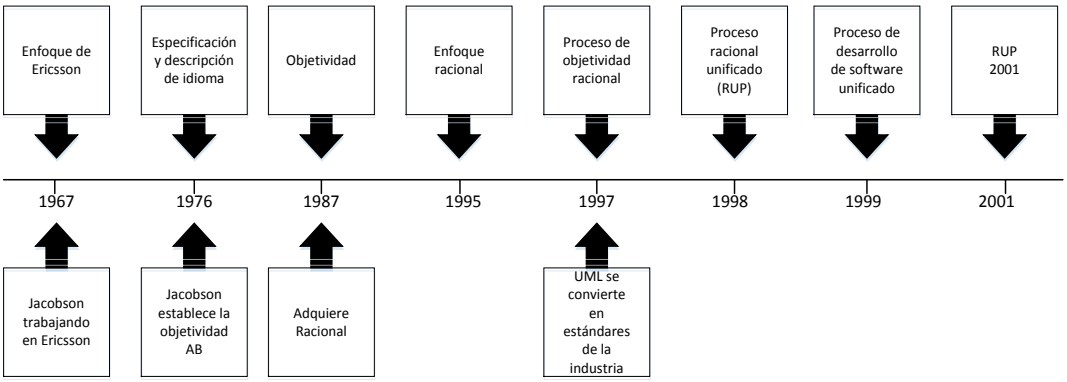

Figura 2. Historia del RUP
Fuente: Jacobson, Booch y Rumbaugh, 2000

Luego, en el periodo de 1995 Rational Software Corporation, conocido por las siglas ROP, compra a Objectory AB, siendo que durante los periodos entre 1995 y 1997 es desarrollado a partir de Objectory 3.8 y el Enfoque Rational (Rational Approach) utilizando el UML como lenguaje de modelación. A partir de ese momento y bajo la dirección de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software, se desarrollaron e incorporaron múltiples componentes que permitieron la expansión de ROP, sobresaliendo en especial el flujo de trabajo denominado modelado del negocio. Además, durante junio del año 1998, es lanzado el Rational Unified Process.

3.4.2. Características principales

Según Jacobson, Booch y Rumbaugh (2000), los diseñadores de RUP manifiestan que la propuesta de RUP, respecto al proceso de software, consta de tres factores primordiales: dirección por Casos de Uso, centralización en arquitectura, y, además resulta ser incremental e interactivo.

3.4.2.1. Proceso dirigido por casos de uso

Según Jacobson, Booch y Rumbaugh (2000), los casos de uso, constituye la técnica de captura de requisitos, lo cual obliga a idearlo en base a los factores de mayor trascendencia desde la óptica usuario y no en base a las funciones que resultarían convenientes para contemplar. Asimismo, son definidos como aquella porción del funcionamiento del sistema que se encarga de dotar del valor agregado para el usuario, en pocas palabras, podemos definirlos como los requisitos funcionales del sistema. En RUP los Casos de Uso no han de ser considerado solamente como una herramienta que precisa o detalla los requisitos de sistema; sino, a su vez son guía del diseño, implementación y prueba. Es decir, que son el factor que integra y orienta el trabajo, conforme se revela en la figura 3.

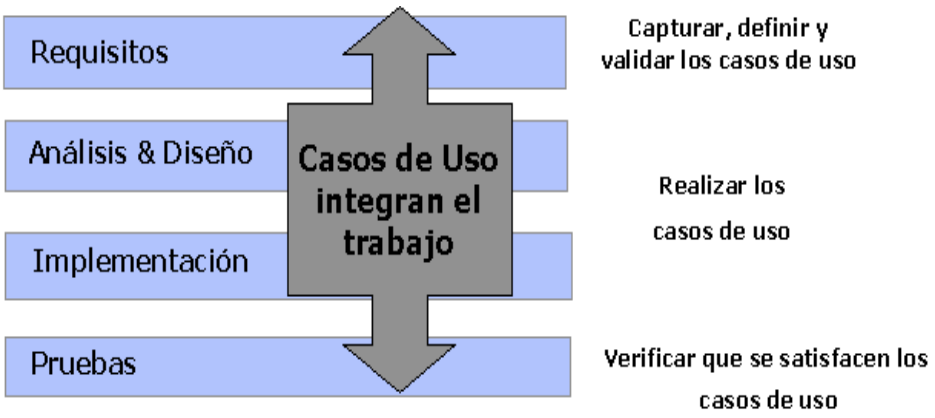

Figura 3. Los Casos de Uso integran el trabajo
Fuente: Jacobson, Booch y Rumbaugh, 2000

Los Casos de Uso son con los que se comienza a desarrollar y además, otorgan el hilo conductor, posibilitando el establecimiento de trazo entre cada artefacto que sea originado entre las actividades del proceso de desarrollo. La figura 4 revela el trazo al que hacíamos referencia en el párrafo anterior; pues en base a Casos de Uso es que se desarrollan los modelos de análisis y diseño,

posteriormente la implementación necesaria para su realización, la verificación efectiva de que el producto haya implementado de forma adecuada cada uno de los Casos de Uso, debiendo encontrarse cada modelo en sincronía.

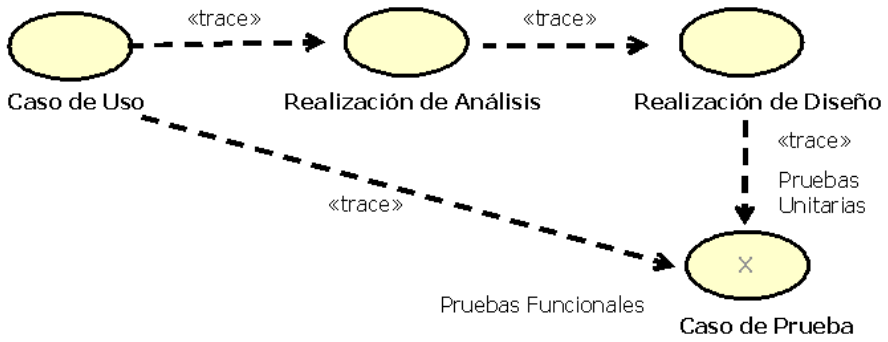

Figura 4. Trazabilidad a partir de los Casos de Uso
Fuente: Jacobson, Booch y Rumbaugh, 2000

3.4.2.2. Procesamiento focalizado en la arquitectura

Según Jacobson, Booch y Rumbaugh (2000), como arquitectura de un determinado sistema, podemos entender a aquella organización o estructura correspondiente a las partes más importantes, la cual posibilitará obtener la visión común de la totalidad de involucrado, es decir, desarrollador y usuario, así como la óptica nítida de sistema completado, la cual es fundamental para el control del desarrollo. La arquitectura es la encargada de relacionar los estáticos y dinámicos de mayor importancia en el sistema, encontrándose íntimamente relacionada a la toma de decisiones que determinan la forma en que debe construirse el sistema y contribuyen a fijar el orden adecuado. Asimismo, la determinación de arquitectura deberá considerar los componentes referidos a la calidad con la deberá contar el sistema, su rendimiento, reutilización y capacidad de evolución; considerando esto, se requiere de flexibilidad en todas las etapas del desarrollo. La determinación de la arquitectura se verá sujeta a las características particulares de

la plataforma, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo (sistemas heredados), varias de las menciones son requisitos no funcionales propios del sistema.

El RUP, adicionalmente deberá centrarse en establecer oportunamente la arquitectura adecuada que no pueda ser radicalmente influenciada por las modificaciones que se realicen con posterioridad a las etapas de construcción y mantenimiento. Los productos poseen forma y función. La función se refiere al funcionamiento que ha de reflejado en los Casos de Uso; mientras que la forma, es la es determinada en base a la arquitectura. La intervención entre ambos consiste en que; Casos de Uso deberán acoplarse a la arquitectura en el momento de su realización, y la arquitectura deberá propiciar el desarrollo de estos, tanto aquellos que son empleados en la actualidad como los que resulten necesarios en el futuro. Lo cual, propicia la evolución paralela de ambos a lo largo del proceso de desarrollo.

La figura 5 muestra el progreso de la arquitectura en cada una de las fases RUP. Donde, se revela que la a arquitectura es más robusta durante las últimas etapas; mientras que, en las primeras únicamente se consolidad a través de baselines y su modificación depende de los requerimientos del proyecto.

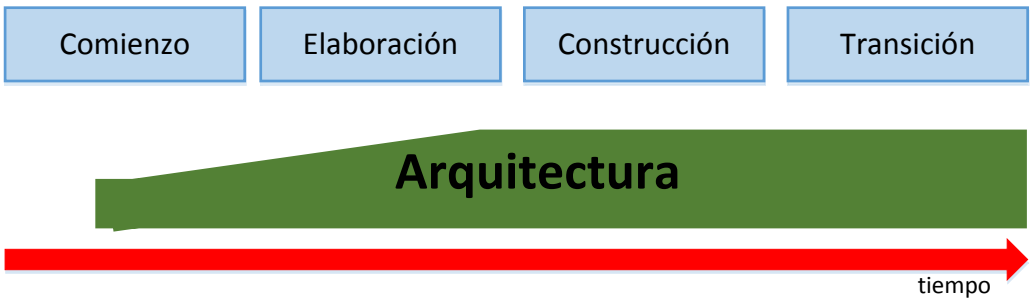

Figura 5. Evolución de la arquitectura del sistema
Fuente: Jacobson, Booch y Rumbaugh, 2000

Resulta fundamental observar el sistema a partir de diversas ópticas, a fin de comprender con mayor exactitud su diseño, motivo por el cual la arquitectura cuenta con representaciones en diferentes vistas centradas en cada aspecto concreto del sistema, distinguiéndose del resto. Para RUP, la totalidad de las vistas conjuntas responden a la denominación de modelo 4+1 de la arquitectura, el mismo que se encuentra conformados por todas las vistas: lógica, de implementación, de proceso y despliegue, adicionalmente a los de Casos de Uso que son los que las unen.

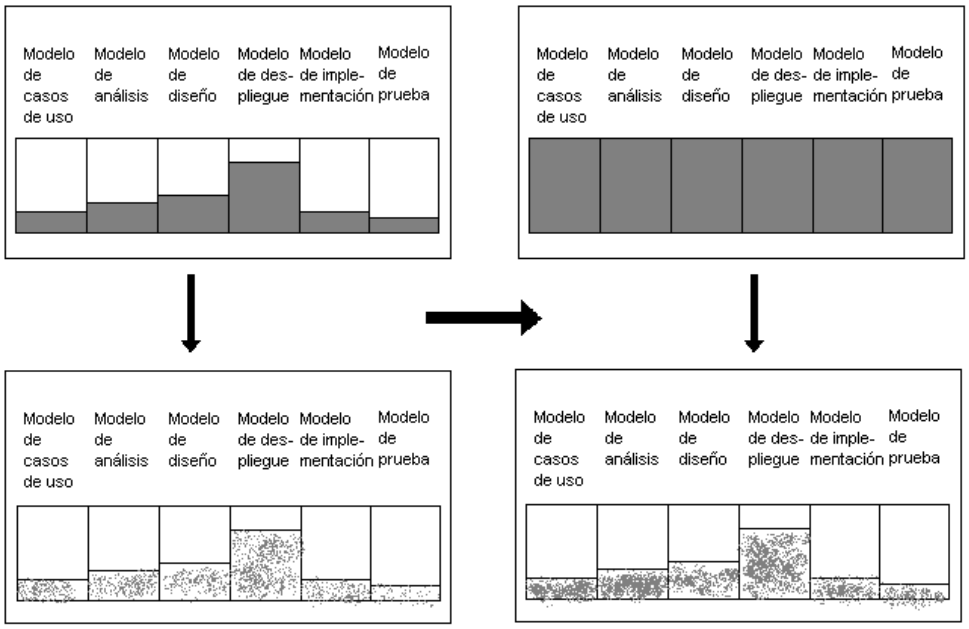

Figura 6. Los modelos se completan, la arquitectura no cambia
Fuente: Jacobson, Booch y Rumbaugh, 2000

Al momento de finalizar la etapa de elaboración se obtendrá como resultado una baseline de la arquitectura, en la que se encontrarán los Casos de Uso que fueron seleccionados en base a su mitigación de riesgos más trascendentales, fundamentales para el usuario y aquellos que se encarguen de las funciones más importantes; por tanto, son considerados como los más relevantes.

Podemos visualizar en la figura 6, que, a lo largo de la construcción de los distintos modelos, estos son desarrollados hasta culminarse, conforme puede observarse en las figuras rellena de la parte superior derecha. Sin embargo, la descripción respecto a la arquitectura, no es sujeto de modificaciones sustanciales como observamos en la parte inferior derecha, debido a que gran parte de la arquitectura fue definida en la elaboración. Asimismo, las modificaciones incorporadas son mínimas respecto a la arquitectura y estos pueden ser visualizados en la densidad de puntos del recuadro que se encuentra en la parte de abajo al lado derecho.

3.4.2.3. Proceso iterativo e incremental

Según Jacobson, Booch y Rumbaugh (2000), como hemos analizado anteriormente la relación entre los Casos de Uso y la arquitectura, para alcanzar su equilibrio adecuado, deberá tomarse en consideración la forma y función durante el desarrollo del producto. Por tanto, la táctica propuesta en RUP consiste en realizar el proceso de forma iterativa e incremental; debiendo el trabajo dividirse en pequeñas partes o mini proyectos. Posibilitando de esa manera el equilibrio y la consecución en cada división realizada y de esa forma para el proceso de desarrollo en su integridad. Cada división puede ser observada a partir de las interacciones, aquel recorrido que se presenta durante los flujos de trabajo fundamentales, obteniéndose un adicional que produce un exponencial crecimiento del producto.

Cada iteración podría ser realizada a través de una cascada, conforme se observa en la figura 7 donde, recorre los flujos esenciales, como son los requisitos, análisis, diseño, implementación y pruebas. Además, se ha de

considerar las actividades de planificar, analizar y algunas otras propias de la iteración. Finalmente, se integran los resultados con el producto de las iteraciones anteriores.

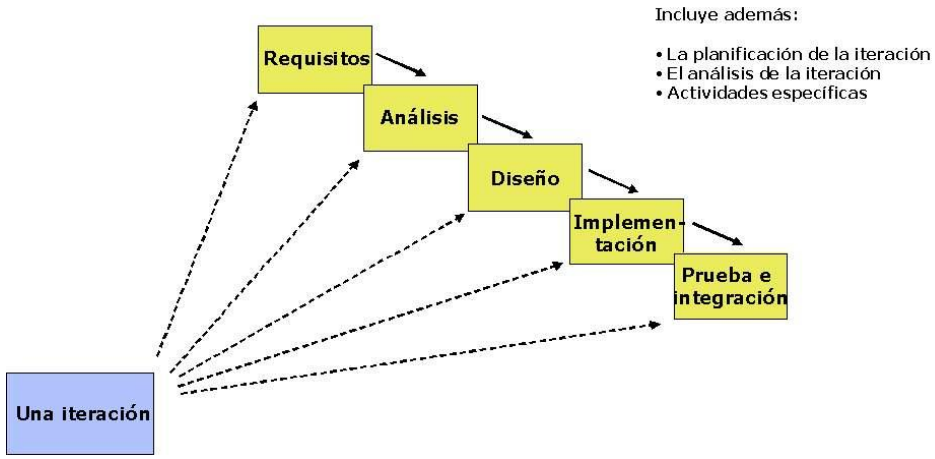

Figura 7. Una iteración RUP
Fuente: Jacobson, Booch y Rumbaugh, 2000

El procesamiento iterativo e incremental se encuentra conformado por la secuencia de iteraciones. Donde, cada una es parte representativa del funcionamiento integro, recorriendo desde los flujos de trabajo trascendentales y ajustando la arquitectura; debiendo ser analizadas al momento de su finalización. Resultando posible su determinación respecto al surgimiento de nuevos requisitos o la modificación de los existentes, incidiendo en las iteraciones siguientes. A lo largo de la planificación de detalles de la siguiente iteración, el equipo deberá investigar como incidirán los riesgos que quedan pendientes durante el trabajo en desarrollo. Las retroalimentaciones de iteraciones anteriores permitirán efectuar reajustes a los objetivos para las futuras iteraciones, debiendo repetirse esta dinámica hasta la culminación total del producto. RUP considera que el proceso se encuentra conformado por cuatro fases, en las que efectúan diversidad de iteraciones en forma indefinida dependiendo del proyecto y en las que se efectúa

mayor o menor hincapié en las diversas actividades. La figura 8 claramente refleja las variaciones de esfuerzos asociados o relacionados a las distintas disciplinas, en función a la fase en que se encuentra el proyecto.

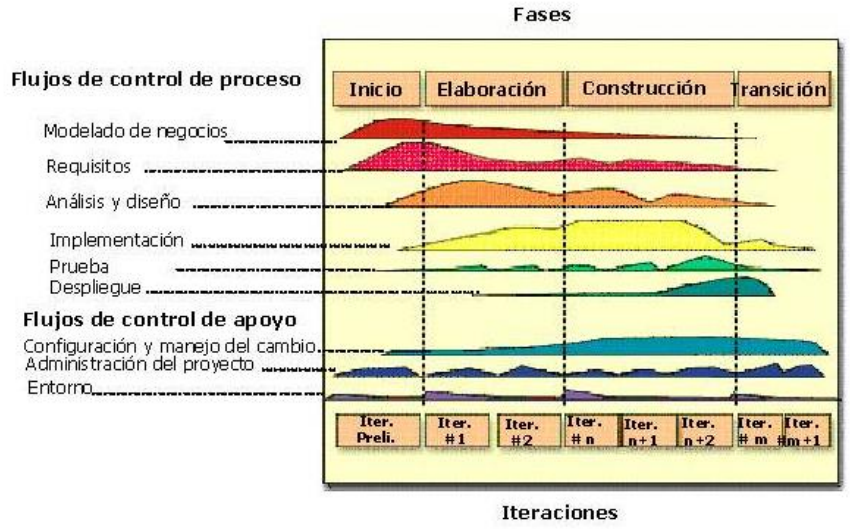

Figura 8. Esfuerzo en actividades según fase del proyecto
Fuente: Jacobson, Booch y Rumbaugh, 2000

Las iteraciones iniciales, es decir, las realizadas durante las dos primeras fases; son basadas en la comprensión del problema y la tecnología, la delimitación del ámbito, supresión de riesgos críticos, y establecimiento de baseline de arquitectura. En la primera fase (inicio), las iteraciones se focalizan en aquellas actividades referidas a la modelación del negocio y los requisitos.

Durante la segunda fase (elaboración), las iteraciones se focalizan a desarrollar la baseline de la arquitectura, comprendiendo los flujos de trabajo de requerimientos, refinamiento del modelo de negocios, análisis, diseño y parte de la implementación orientada a la baseline. Durante la tercera fase (construcción), se realiza la construcción del producto mediante un conjunto de iteraciones.

Por cada iteración se deberá elegir algún Caso de Uso, revisando previamente su análisis y diseño y su consecuente implementación y pruebas. Se efectúa una pequeña cascada por cada ciclo, así como la ejecución de aquellas

iteraciones que resulten necesarias hasta finalizar la implementación de la nueva versión del producto. Finalmente, en la cuarta fase (transición) se pretende garantizar que el producto se encuentra finalizado y listo para ser entregado a los usuarios.

3.4.3. Otras prácticas

Según Jacobson, Booch y Rumbaugh (2000), RUP reconoces seis prácticas como las mejores, mediante las cuales determina la forma efectiva en la que deberán trabajar los equipos de desarrollo de software.

3.4.3.1. Administración de requisitos.

Según Jacobson, Booch y Rumbaugh (2000), proporciona orientación mediante una guía que permite hallar, establecer, registrar y continuar con los cambios de requisitos sean funcionales o restricciones; además, emplea una notación de Caso de Uso y escenarios que permiten la representación de requisitos.

3.4.3.2. Desarrollo de software iterativo.

Según Jacobson, Booch y Rumbaugh (2000), implica desarrollar el producto a través de iteraciones con hitos claramente determinados, donde, las actividades son repetidas con diferente énfasis, en función a la fase en que se encuentre.

3.4.3.3. Desarrollo basado en componentes.

Según Jacobson, Booch y Rumbaugh (2000), para la invención de sistemas intensivos, se requerirá previamente partir el sistema por componentes, donde las interfaces ya deberán estar determinadas y luego deberán ser ensamblados a fin de generar el sistema. A lo largo del desarrollo, permite que el sistema se cree conforme a son obtenidos o desarrollados los componentes.

3.4.3.4. Modelado visual (usando UML).

Según Jacobson, Booch y Rumbaugh (2000), UML es un lenguaje para visualizar, determinar, erigir y registrar artefactos de un sistema software. Para la OMG, es un estándar el emplear herramientas de modelado visual porque facilitan la administración de los modelos, además que encubren o explican los detalles de ser necesario. Además, este tipo de modelado permite mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. En pocas palabras, contribuye al mejoramiento de la capacidad del equipo para la administración de la complejidad del software.

3.4.3.5. Verificación continua de la calidad.

Según Jacobson, Booch y Rumbaugh (2000), resulta fundamental garantizar la calidad de la totalidad de artefactos, por ello deben ser evaluados en distintos puntos a lo largo de su desarrollo, dando mayor énfasis al culminar cada iteración. En la ejecución de las verificaciones, las pruebas cumplen un rol protagónico y por ello han de ser incorporados en el proceso. Además, en el caso de artefactos no ejecutables, de igual forma se deberán realizar revisiones e inspecciones de forma continua.

3.4.3.6. Gestión de los cambios.

Según Jacobson, Booch y Rumbaugh (2000), los cambios son considerados factores de riesgo de carácter crítico para el desarrollo de proyectos. Debido a que los artefactos son modificados permanentemente por las actividades de mantenimiento que se efectúan con posterioridad a la entrega del producto, y además a lo largo del desarrollo, resultando ser fundamentales por el probable efecto que generen sobre los requisitos. Adicionalmente, debe considerarse el

desafío que implica la construcción de un software con la intervención de diversos desarrolladores, los cuales sin importar su distribución geográfica pueden estar trabajando mediante una reléase o diversas plataformas. Es así que, la falta de disciplina podría conducirlos de forma rápida al caos, por ello esta disciplina se encarga de este aspecto tan fundamental.

3.4.4. Estructura del proceso

Según Jacobson, Booch y Rumbaugh (2000), se puede describir el proceso a partir de dos ejes:

3.4.4.1. *Eje horizontal.*

Es aquel que, plasma de forma representativa el tiempo y se le considera como el eje de aspectos dinámicos. Señala los detalles propios del ciclo de vida del proceso en términos de fases, iteraciones e hitos. En la figura 9, podemos observar las fases del RUP: Inicio, Elaboración, Construcción y Transición. Conforme mencionamos en líneas anteriores, cada una, se encuentra conformada por iteraciones.

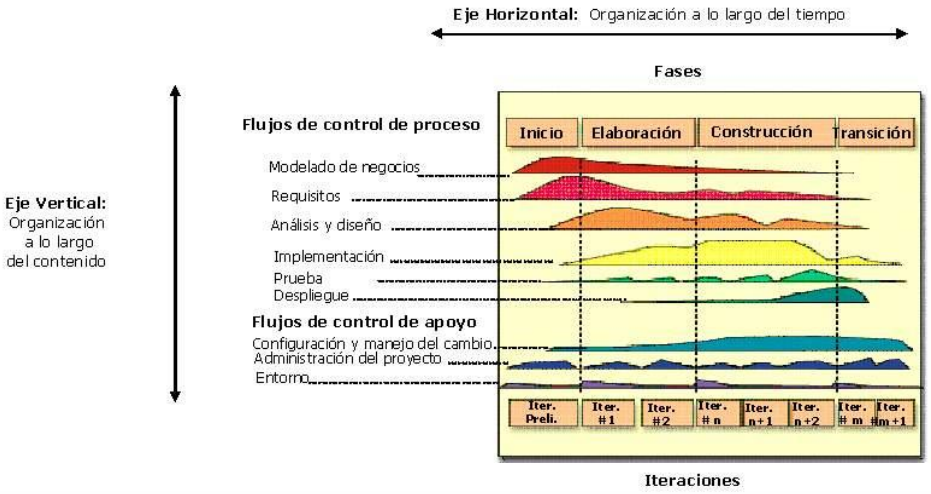

Figura 9. Estructura de RUP
Fuente: Jacobson, Booch y Rumbaugh, 2000

3.4.4.2. Eje vertical.

Simboliza los criterios estáticos; describiendo el proceso con base a los componentes de proceso, disciplinas, flujos de trabajo, actividades, artefactos y roles.

3.4.5. Estructura dinámica del proceso, fases e iteraciones

Según Jacobson, Booch y Rumbaugh (2000), el RUP es repetido durante los ciclos que integran la vida de un producto. En la figura 10, cada uno de los ciclos, culmina con la obtención producto para los usuarios; comprendiendo las cuatro fases ya mencionadas. Cada una de estas se encuentra conformada por iteraciones de número variable, dependiendo de la fase en la que se encuentran.

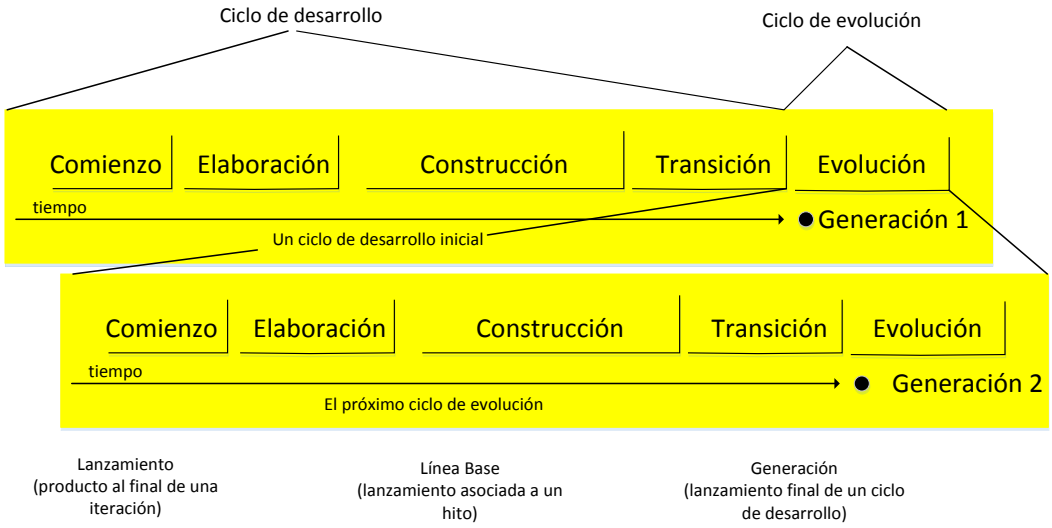

Figura 10. Ciclos, Lanzamientos, Línea Base
Fuente: Jacobson, Booch y Rumbaugh, 2000

Cada una de las fases, culmina con un hito claramente definido, es un punto en el tiempo en el que se deberán tomar determinadas decisiones críticas para poder conseguir las metas claves para poder continuar con la siguiente fase. El hito principal propio de cada una de las fases, se encuentra comprendido por hitos menores que pueden ser los criterios que resultan aplicables a cada iteración.

Los hitos para cada fase son: Inicio – Objetivos del ciclo de vida, Elaboración – Arquitectura del ciclo de vida, Construcción – Capacidad operacional inicial, Transición – Lanzamiento del producto. Es así que, en la figura 11, podemos visualizar cada una de las fases con sus respectivos hitos.

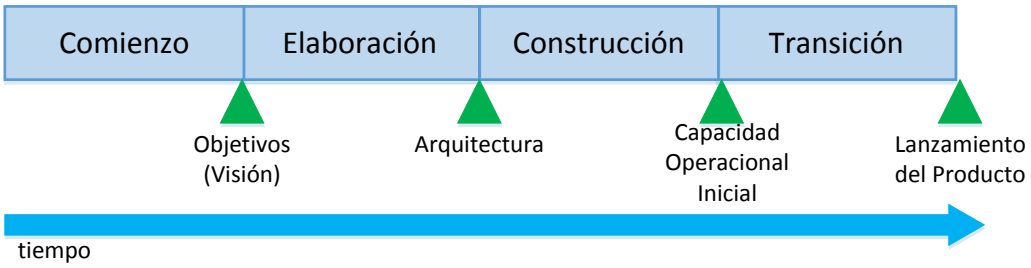

Figura 11. Fases e hitos en RUP
Fuente: Jacobson, Booch y Rumbaugh, 2000

Con bases a las características propias del proyecto, son determinadas o definidas la duración y el esfuerzo que ha de dedicarse a cada fase, resultando variables. La tabla 1 revela los porcentajes más usuales en referencia a lo señalado. En función al esfuerzo señalado, en la figura 12 se muestran las típicas distribuciones de recursos humanos que son necesarios durante el proyecto.

Tabla 1

Distribución típica de esfuerzo y tiempo

	Inicio	**Elaboración**	**Construcción**	**Transición**
Esfuerzo	5 %	20 %	65 %	10 %
Tiempo dedicado	10 %	30 %	50 %	10 %

Fuente: Jacobson, Booch y Rumbaugh, 2000

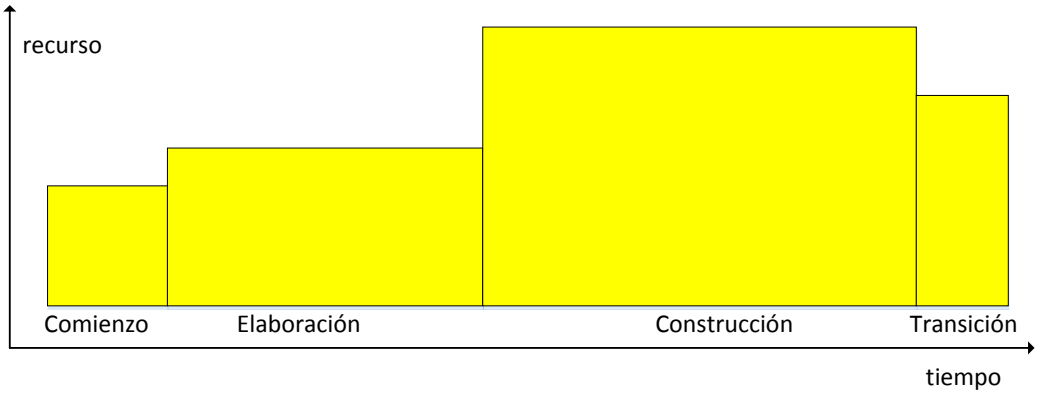

Figura 12. Distribución típica de recursos humanos
Fuente: Jacobson, Booch y Rumbaugh, 2000

3.4.6. Fases de desarrollo RUP

Según Jacobson, Booch y Rumbaugh (2000), se puede identificar las siguientes fases:

3.4.6.1. Inicio.

En esta fase, son determinados el modelo de negocio y su alcance. Además, de identificar a los actores participantes y Casos de Uso, estos últimos son diseñados en un 20 % más o menos pero sólo aquellos más esenciales. El plan de negocio es desarrollado a fin de identificar los recursos que se deberán asignar en el proyecto.

Sus objetivos son:

- Determinar el proyecto, respecto a su ámbito y límites.
- Identificar los Casos de Uso críticos del sistema, así como aquellos escenarios básicos que determinan la funcionalidad.
- Revelar al menos una arquitectura candidata para los escenarios principales.
- Estimar el coste en términos de recursos y tiempo, respecto a la totalidad del proyecto.
- Estimar los riesgos y fuentes de incertidumbre.

Los resultados que se esperan obtener son:

- Un documento de visión: Visión general de los requerimientos del proyecto, características clave y restricciones principales.
- Modelo inicial de Casos de Uso (culminado en 10-20 %).
- Un glosario inicial: Terminología clave del dominio.
- El caso de negocio.
- Lista de riesgos y plan de contingencia.
- Plan del proyecto, mostrando fases e iteraciones.
- Modelo de negocio, de ser necesario
- Prototipos exploratorios para probar conceptos o la arquitectura candidata.

A fin de continuar con el desarrollo del proyecto, en esta se deberán verificar los términos de evaluación:

- A la totalidad de interesados coinciden en la definición del ámbito del sistema y sus estimaciones de agenda.
- Comprensión de requisitos, como evidencia de la fidelidad de los Casos de Uso principales.
- Estimaciones de tiempo, coste y riesgo son creíbles.
- Entendimiento absoluto de cualquier prototipo de la arquitectura desarrollada.
- Gastos son similares a los planificados.

En caso de que, el proyecto no responda a los términos de evaluación señalados, se deberá evaluar con detenimiento sí es conveniente abandonarlo o replantearlo de forma profunda.

3.4.6.2. Elaboración.

La intención de esta fase es examinar el dominio del problema, construir la base de la arquitectura, desarrollar el plan y eliminar los mayores riesgos. En esta, se elabora el prototipo de arquitectura, debiendo evolucionar en las sucesivas iteraciones hasta convertirse en el sistema final. El prototipo deberá estar conformador por Casos de Uso que fueron determinados en la fase anterior y deberá demostrar que los riesgos más graves fueron evitados.

Sus objetivos son:

- Definición, validación y cimentación de la arquitectura.
- Culminación de la visión.
- Creación de plan confiable para la próxima fase. Debiendo perfeccionarse en las iteraciones sucesivas. Debe incluir los costes si procede.
- Demostración de que arquitectura propuesta soporta visión, considerando que su coste y tiempo son razonables.

Los resultados que se esperan obtener, son:

- Modelo de Casos de Uso completo a un 80 %, como mínimo. Los casos y actores deben estar plenamente identificados y los casos se deben encontrar desarrollados en su mayoría.
- Requisitos adicionales que comprenden los no funcionales y aquellos que no se encuentran asociados con algún Caso de Uso en específico.
- Descripción de la arquitectura software.
- Prototipo ejecutable de la arquitectura.
- Listado de riesgos y caso de negocio revisados.
- Plan de desarrollo del proyecto.

- Caso de desarrollo actualizado que detalla el proceso a seguir.
- Opcionalmente, manual de usuario preliminar.

Durante esta fase, se trata de abarcar el proyecto en su totalidad, pero de forma superficial, es decir a un nivel de profundidad mínimo, pues sólo se verán con mayor intensidad los riesgos más críticos e importantes de la arquitectura. Además, se actualizarán todos los productos resultantes de la primera fase.

En esta fase, se consideran como términos de evaluación:

- Visión del producto estable.
- Arquitectura estable.
- Demostración de que principales elementos de riesgos fueron abordados y resueltos, mediante la ejecución del prototipo.
- El plan de construcción es detallado y preciso, cuenta con estimaciones creíbles.
- La totalidad de interesados coinciden en que la visión actual podrá ser lograda sí se cumplen los planes actuales en el contexto de la arquitectura actual.
- Los gastos realizados hasta esta fase se consideran aceptables en comparación con lo planificado.

De igual forma que en la fase anterior, de considerarse que el proyecto no cumple con los términos de evaluación mencionados, se debe evaluar la continuación del proyecto, considerando sí es necesario abandonarlo o replantearlo de forma considerable.

3.4.6.3. Construcción.

Esta tiene por objeto principal, conseguir la máxima capacidad operacional del producto mediante las iteraciones sucesivas. A lo largo de la realización de esta fase, se deben implementar, integras y probar la totalidad de componentes, características y requisitos; hasta que se obtenga una versión aceptable del producto.

Concretamente, sus objetivos son:

- Minimización de costes de desarrollo, a través de la optimización de recursos y evitando rehacer o deshacer el trabajo.
- Calidad adecuada en el menor tiempo posible.
- Obtención de versiones funcionales (alfa, beta, y otras versiones de prueba) en el menor tiempo posible.

En esta fase, se espera obtener los siguientes resultados:

- Modelos completos (Casos de Uso, Análisis, Diseño, Despliegue e Implementación).
- Arquitectura íntegra (mantenida y mínimamente actualizada).
- Riesgos presentados mitigados.
- Plan del proyecto para la fase de transición.
- Manual inicial de usuario (con suficiente detalle).
- Prototipo operacional – beta.
- Caso del negocio actualizado.

Los criterios de evaluación de esta fase son los siguientes:

- Producto estable y maduro que puede ser entregado a usuarios para ser probado.

- La totalidad de usuarios expertos se encuentran listos para ser parte de la comunidad de usuarios.
- La diferencia entre los gastos planificados y ejecutados es aceptable.

3.4.6.4. Transición.

Esta fase tiene por objeto la disposición del producto a los usuarios finales, debiendo previamente desarrollarse versiones actualizadas del producto, terminar la documentación, preparar al usuario para manejar el producto; y la realización de todas aquellas tareas vinculadas al ajuste, configuración, instalación y facilidad de uso del producto.

En esta fase, se podrán incluir:

- Prueba de la versión beta para validar el nuevo sistema frente a las expectativas de los usuarios.
- Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por nuestro proyecto.
- Conversión de las bases de datos operacionales.
- Entrenamiento de los usuarios y técnicos de mantenimiento.
- Transferencia del producto a los equipos de marketing, distribución y venta.

Sus objetivos fundamentales son:

- Lograr que el usuario se valga por sí mismo.
- Producto final cumple con requisitos planificados, funciona y satisface al usuario.

En esta fase se espera obtener los siguientes resultados:

- Prototipo operacional.
- Documentos legales.

- Caso del negocio completo
- Línea de base del producto completa y corregida que incluye todos los modelos del sistema.
- Descripción de la arquitectura (total y corregida).
- Las iteraciones se encuentran focalizadas en la obtención de versión nueva.

Los términos para realizar la evaluación en esta fase, son:

- Usuario satisfecho.
- Diferencia entre gastos planificados y ejecutados, es aceptable.

3.4.7. Estructura estática del proceso

Según Jacobson, Booch y Rumbaugh (2000), se puede identificar la siguiente estructura:

3.4.7.1. Roles, actividades, artefactos y flujos de trabajo.

En las figuras 13 y 14 se muestra los procesos de desarrollo de software dan respuesta a las incógnitas de: quién, cómo, qué y cuándo. RUP, determinar los roles para la primera pregunta; las actividades que comprenden la segunda; y los productos que han de obtener en la tercera y, finalmente los flujos de trabajo de disciplinas que dan respuesta a la última incógnita.

Figura 13. La conexión entre roles, actividades, artefactos
Fuente: Jacobson, Booch y Rumbaugh, 2000

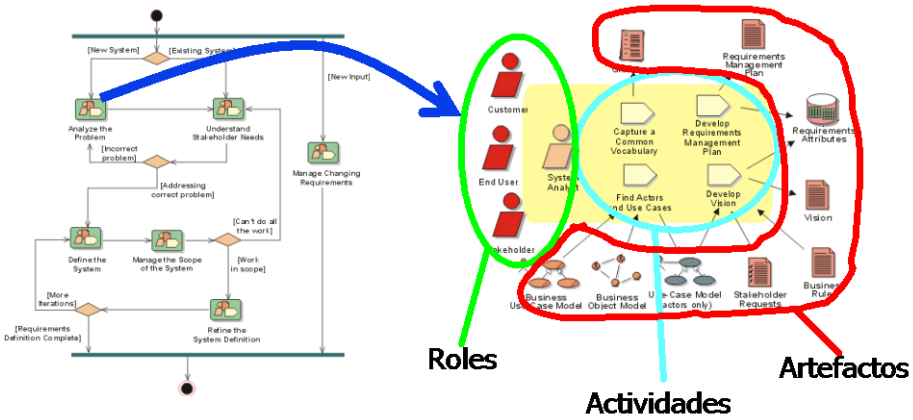

Figura 14. Descripción de un workflow a través de roles, actividades y artefactos
Fuente: Jacobson, Booch y Rumbaugh, 2000

a. *Roles.*

Los roles son los que definen el comportamiento y conjunto de responsabilidades del individuo, o de un conjunto de estos que se desempeñan como equipo. Una misma persona puede ejecutar varios roles, de igual forma, un rol podrá ser desempeñado por diversas personas. La responsabilidad de ejecutar un rol determinado, implica la realización de un grupo de tareas, así como ser propietario de varios artefactos.

El RUP agrupa los roles según la participación que efectúan en actividades relacionadas, resultando de tal agrupación, los siguientes grupos:

a.1. Analistas.

- Analista de procesos de negocio.
- Diseñador del negocio.
- Analista de sistema.
- Especificador de requisitos.

a.2. Desarrolladores.

- Arquitecto de software.
- Diseñador

- Diseñador de interfaz de usuario
- Diseñador de cápsulas.
- Diseñador de base de datos.
- Implementador.
- Integrador.

a.3. Gestores.

- Jefe de proyecto
- Jefe de control de cambios.
- Jefe de configuración.
- Jefe de pruebas
- Jefe de despliegue
- Ingeniero de procesos
- Revisor de gestión del proyecto
- Gestor de pruebas.

a.4. Apoyo.

- Documentador técnico
- Administrador de sistema
- Especialista en herramientas
- Desarrollador de cursos
- Artista gráfico

a.5. Especialista en pruebas.

- Especialista en pruebas (tester)
- Analista de pruebas
- Diseñador de pruebas

a.6. Otros roles.

- Stakeholders.
- Revisor
- Coordinación de revisiones
- Revisor técnico
- Cualquier rol

b. Actividades.

Es la unidad de trabajo que una persona que desempeñada un rol puede ser requerido a que realice; su objeto versa en la creación o actualización de un determinado producto.

c. Artefactos.

También son denominados como productos, consistente en la porción de información que se produce, modifica o usar en el desarrollo del software; constituyen el resultado tangible del proyecto, así como lo que se emplea, usa y crea a fin de culminar el producto. Son considerados como artefactos, los siguientes:

- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de Casos de Uso o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un Caso de Uso o un subsistema.

d. *Flujos de trabajo.*

Con la enumeración de roles, actividades y artefactos no se define un proceso, necesitamos contar con una secuencia de actividades realizadas por los diferentes

roles, así como la relación entre los mismos. Un flujo de trabajo es una relación de actividades que nos producen unos resultados observables. A continuación se dará una explicación de cada flujo de trabajo.

d.1. Modelado del negocio.

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el producto. Los objetivos del modelado de negocio son:

- Comprender la estructura y la dinámica de la organización para la cual el sistema va ser explicado (organización objetivo).
- Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.
- Derivar los requisitos del sistema necesarios para apoyar a la organización objetivo.

Para lograr estos objetivos, el modelo de negocio describe como desarrollar una visión de la nueva organización, basado en esta visión se definen procesos, roles y responsabilidades de la organización por medio de un modelo de Casos de Uso del negocio y un modelo de objetos del negocio. Complementario a estos modelos, se desarrollan otras especificaciones tales como un glosario.

d.2. Requisitos.

Este es uno de los flujos de trabajo más importantes, porque en él se establece qué tiene que hacer exactamente el sistema que construyamos. En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales

tienen que comprender y aceptar los requisitos que especifiquemos. Los objetivos de los requisitos son:

- Establecer y mantener un acuerdo entre clientes y otros stakeholders sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.
- Definir el ámbito del sistema.
- Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- Proveer una base para estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

d.3. Análisis y diseño.

El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema. Los objetivos del análisis y diseño son:

- Transformar los requisitos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento.

Otro producto importante de este flujo es la documentación de la arquitectura de software, que captura varias vistas arquitectónicas del sistema.

d.4. Implementación.

En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. Además se deben hacer las pruebas de unidad: cada implementador es responsable de probar las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable. La estructura de todos los elementos implementados forma el modelo de implementación. La integración debe ser incremental, es decir, en cada momento sólo se añade un elemento. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir el riesgo. Su utilidad puede ir desde ver si el sistema es viable desde el principio, probar tecnologías o diseñar la interfaz de usuario. Los prototipos pueden ser exploratorios (desechables) o evolutivos. Estos últimos llegan a transformarse en el sistema final.

d.5. Pruebas.

Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida. Esta disciplina brinda soporte a las otras disciplinas. Sus objetivos son:

- Encontrar y documentar defectos en la calidad del software.
- Generalmente asesora sobre la calidad del software percibida.
- Provee la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- Verificar las funciones del producto de software según lo diseñado.
- Verificar que los requisitos tengan su apropiada implementación.

d.6. Despliegue.

El objetivo de este flujo de trabajo es producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

- Probar el producto en su entorno de ejecución final.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.
- Formar a los usuarios y al cuerpo de ventas.
- Migrar el software existente o convertir bases de datos.

d.7. Gestión del proyecto.

La gestión del proyecto es el arte de lograr un balance al gestionar objetivos, riesgos y restricciones para desarrollar un producto que sea acorde a los requisitos de los clientes y los usuarios. Los objetivos de este flujo de trabajo son:

- Proveer un marco de trabajo para la gestión de proyectos de software intensivos.
- Proveer guías prácticas realizar planeación, contratar personal, ejecutar y monitorear el proyecto.
- Proveer un marco de trabajo para gestionar riesgos.

3.4.8. Una configuración RUP para proyecto pequeño

Según Jacobson, Booch y Rumbaugh (2000), en este apartado se describe una posible configuración de RUP para un proyecto pequeño. Por las características del proyecto, se han incluido muy pocos artefactos, roles y actividades de la

metodología, manteniendo los más esenciales. Dicha configuración está basada en la siguiente selección de artefactos:

3.4.8.1. Entregables del proyecto.

A continuación, se describen brevemente cada uno de los artefactos que se generarán y usarán durante el proyecto.

3.4.8.2. Flujos de trabajo.

Se utilizarán diagramas de actividad para modelar los flujos de trabajo (workflows) del área problema, tanto los actuales (previos a la implantación de nuevo sistema) como los propuestos, que serán soportados por el sistema desarrollado.

3.4.8.3. Características del producto software.

Es una lista de las características principales del producto, deseables desde una perspectiva de las necesidades del cliente.

3.4.8.4. Glosario.

Es un documento que define los principales términos usados en el proyecto. Permite establecer una terminología consensuada.

3.4.8.5. Modelo de Casos de Uso.

El modelo de Casos de Uso presenta la funcionalidad del sistema y los actores que hacen uso de ella. Se representa mediante diagramas de casos de uso.

3.4.8.6. Especificaciones de Casos de Uso.

Para los casos de uso que lo requieran (cuya funcionalidad no sea evidente o que no baste con una simple descripción narrativa) se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen: precondiciones, pos condiciones, flujo de eventos, requisitos no-funcionales asociados.

3.4.8.7. Modelo de análisis y diseño.

Este modelo establece la realización de los casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño (incluyendo una orientación hacia el entorno de implementación). Está constituido esencialmente por un diagrama de clases y algunos diagramas de estados para las clases que lo requieran.

3.4.8.8. Modelo lógico relacional.

Previendo que la persistencia de la información del sistema será soportada por una base de datos relacional, este modelo describe la representación lógica de los datos persistentes, de acuerdo con el enfoque para modelado relacional de datos. Para expresar este modelo se utiliza un diagrama de tablas donde se muestran las tablas, claves, etc.

3.4.8.9. Modelo de implementación.

Este modelo es una colección de componentes y los subsistemas que los contienen. Estos componentes incluyen: ficheros ejecutables, ficheros de código fuente, y todo otro tipo de ficheros necesarios para la implantación y despliegue del sistema.

3.4.8.10. Modelo de pruebas.

Para cada caso de uso se establecen pruebas de aceptación que validarán la correcta implementación del caso de uso. Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba, y los resultados esperados.

3.4.8.11. Manual de instalación.

Este documento incluye las instrucciones para realizar la instalación del producto.

3.4.8.12. Material de usuario.

Corresponde a un conjunto de documentos y facilidades de uso del sistema.

3.4.8.13. Producto.

Todos los ficheros fuente y ejecutable del producto.

3.4.8.14. Esquema de trazabilidad.

La figura 15 ilustra las relaciones de trazabilidad entre artefactos del proyecto, y según la configuración antes mencionada.

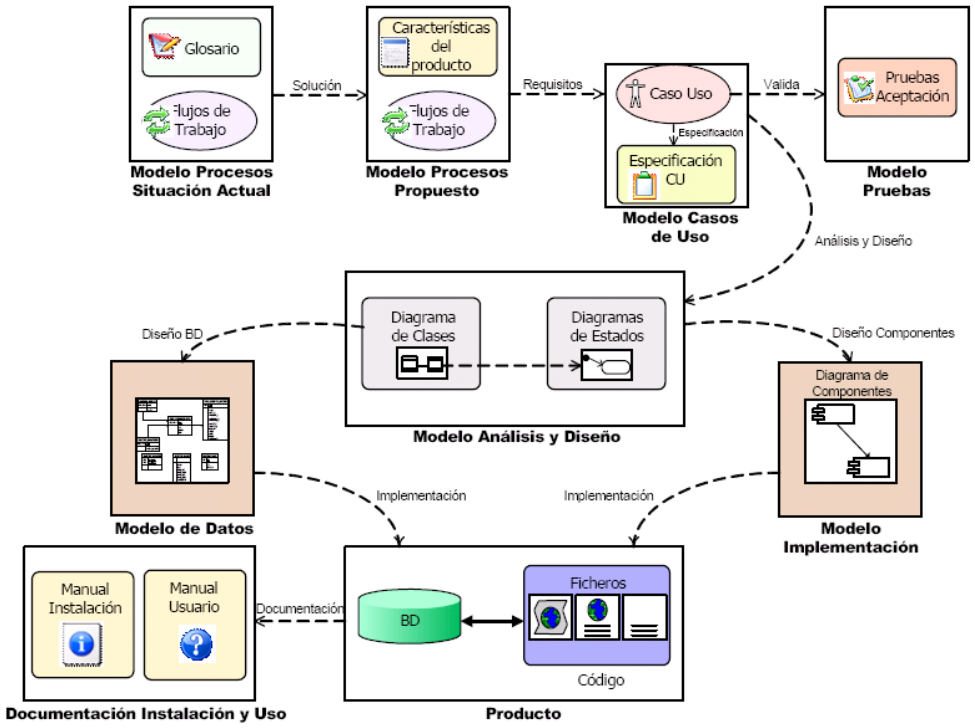

Figura 15. Trazabilidad entre artefactos.
Fuente: Jacobson, Booch y Rumbaugh, 2000

3.5. Caso práctico

En la propuesta tecnológica, se ha desarrollado un sistema de atención de pacientes en la clínica utilizando la tecnología Java y SQL, el cual nos brindará una escalabilidad y performance adecuada. Para este propósito esquematizaremos el procedimiento de atención de los pacientes de la clínica.

Se realizarán las siguientes actividades:

- Recolección de información preliminar, en este caso utilizaremos las técnicas de: documentación, entrevistas y observación directa.
- Verificación de documentación de procesos de la clínica dental Ebenezer.
- Entrevistas programadas, para absolver los requerimientos para el sistema de la clínica dental Ebenezer.
- Elaboración de esquemas preliminares de casos de uso y esquema técnico de la clínica dental Ebenezer.
- Elaboración de diagrama de arquitectura y ver el estado situacional de la implantación del módulo del sistema de gestión de la clínica dental.

En la figura 16 se muestra como se obtiene la información respectiva a fin de determinar si es factible la implementación de un nuevo sistema.

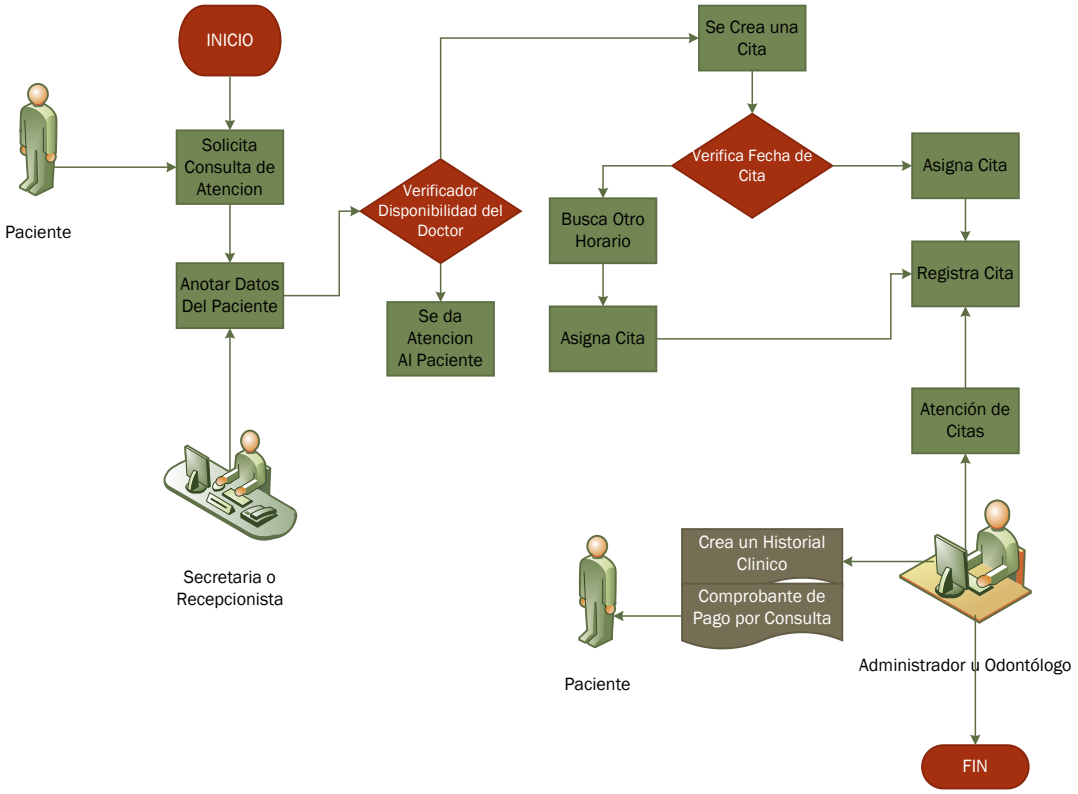

Figura 16. Esquema del proceso de atención de la clínica.

3.5.1. Oportunidad de negocio

Este análisis le permitirá a la clínica dental Ebenezer, desarrollar de una forma adecuada el control de citas, pagos y mejorar la atención al paciente en base a los diagramas establecidos en este documento. El cual mostrará la secuencia de los objetos que intervienen para el desarrollo del sistema y su debida escalabilidad.

Asimismo, el análisis de sistema permitirá construir un sistema con las especificaciones propias de la empresa y permitirá mediante el manejo de indicadores y estadísticas la toma de decisión referente en aspectos propios de la Institución. A continuación, se muestra la figura 17 en el que se muestra la oportunidad de negocio en el manejo de la información.

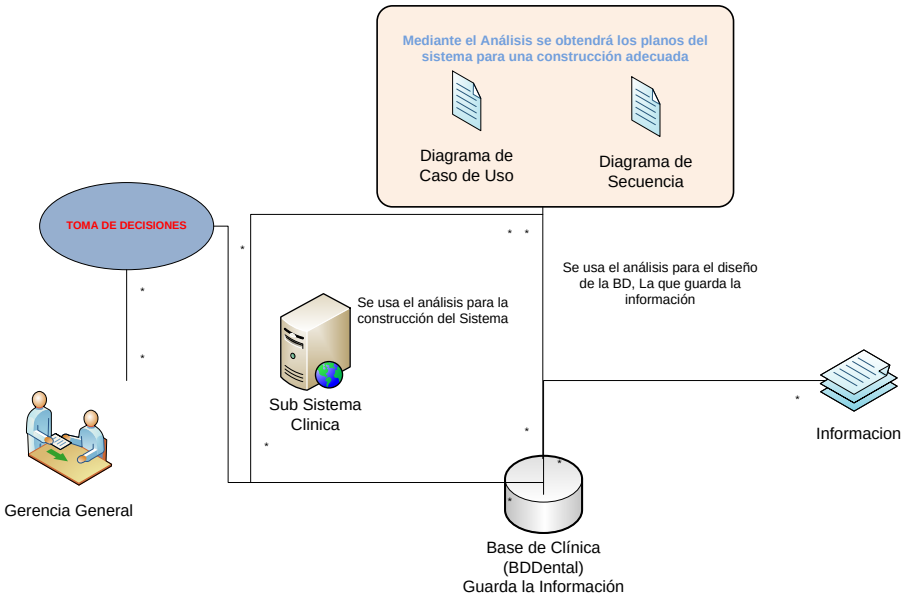

Figura 17. Diagrama de negocio

3.5.2. Definición de requerimientos de información

Se obtiene la información respectiva a fin de conocer nuestro negocio y establecer las reglas de negocio obteniendo así los distintos requerimientos del usuario. A la vez se obtuvo los procesos de dicha división, en la tabla 2 se ve los siguientes:

Tabla 2

Documento de procesos de la clínica dental

Código	Nombre del proceso
PRO.CLI-01	Consulta, Asignación de citas
PRO.CLI-02	Atención, Control de citas

Asimismo en las figuras 18 y 19 se muestra los diagramas de procesos para proceder con el análisis requerido en la realización del sistema.

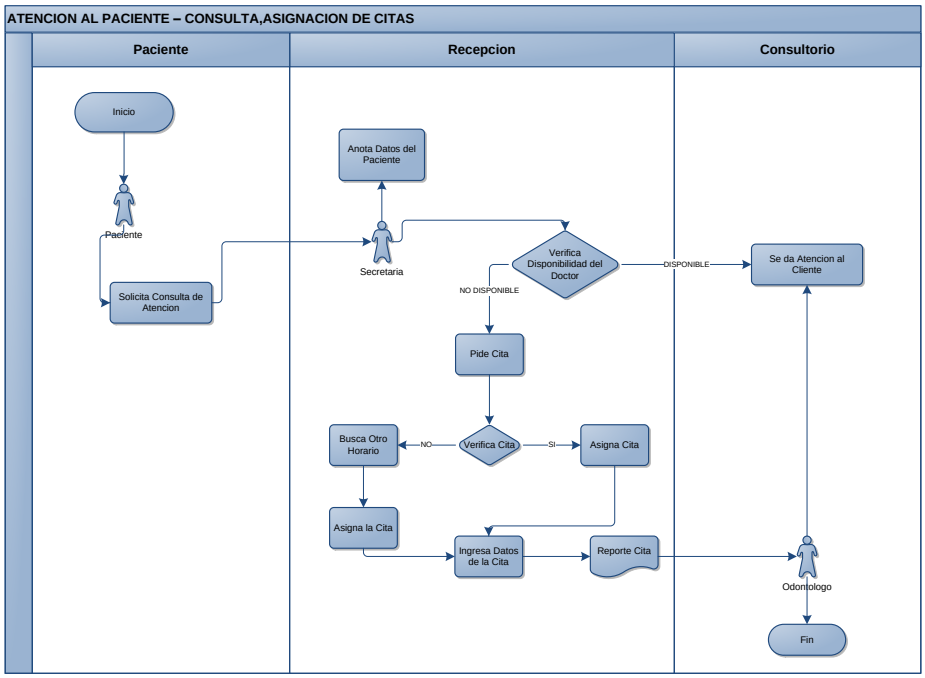

Figura 18. Diagrama de proceso: Consulta, asignación de citas

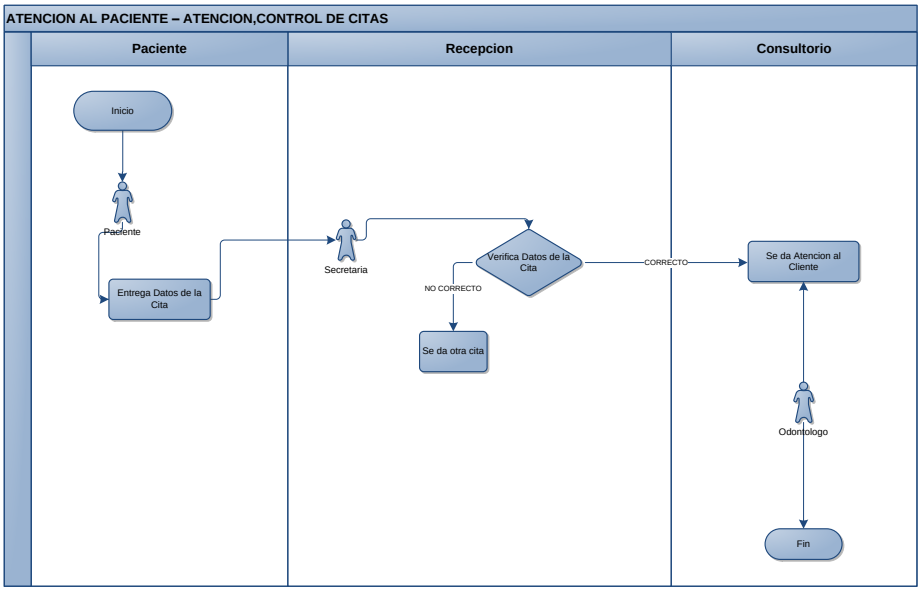

Figura 19. Diagrama de proceso: Atención, control de citas

Se procedió a analizar los diferentes procesos, que se muestran anteriormente. Donde se determinó los procedimientos más importantes que sirven como directriz para la construcción del software. La fase de análisis explicará los procesos más importantes.

3.5.3. Requerimientos para módulos de recepción y administración de la clínica

De acuerdo al levantamiento de información y al analizar la gestión de la clínica dental, se ha determinado los siguientes requerimientos funcionales, ver tabla 3:

Tabla 3
Requerimientos funcionales

Requerimiento	Descripción
REQ-CLI-01	Permitir gestionar el registro de los pacientes que son atendidos dentro de la clínica.
REQ-CLI-02	Permitir gestionar el registro de las citas dadas a los pacientes dentro del área de recepción de la clínica.
REQ-CLI-03	Permitir gestionar el control de pagos de los pacientes por los servicios brindados dentro de la clínica.
REQ-CLI-04	Permitir gestionar el registro de historias clínicas y el odontograma de paciente atendido dentro de la clínica.
REQ-CLI-05	Permitir gestionar el registro del personal que trabaja dentro de la clínica.
REQ-CLI-06	Realizar reportes generales de las citas diarias y semanales, personal y servicios.
REQ-CLI-07	Permitir realizar la impresión de comprobantes de pago,
REQ-CLI-08	Permitir visualizar e imprimir el odontograma de cada paciente

3.5.3.1. Requerimientos no funcionales.

- Contribuir a las capacitaciones permanentes de los usuarios a través de la base de conocimiento.

- Se desarrollará mediante un plan de capacitación que constará de sesiones personales y por grupo de acuerdo al nivel de manejo del software.
- A la vez se le motivará al personal de la institución en el manejo del sistema.
- Se le debe otorgar al personal capacitado: el manual de usuario, el manual de seguridad de cuenta de usuario y las políticas del manejo.

3.5.3.2. Requerimientos del dominio.

- Usuario de línea (nivel 1):
- El usuario podrá acceder a las interfaces, que le permitan realizar gestión sobre la clínica dental Ebenezer: registrar, modificar los datos del personal; registrar, modificar, eliminar los datos e historia clínica del paciente; registrar, modificar, eliminar las citas; consultas por pantalla y/o impresas de acuerdo a requerimiento.
- El usuario podrá acceder a los reportes de gestión inherentes a las actividades que realiza. De acuerdo a los requerimientos definidos, tales como: reporte de las citas, reportes de servicios, reportes del personal.

3.5.4. Análisis del sistema

En el análisis se ha desarrollado diagramas de casos de uso, diagrama de secuencia, arquitectura, despliegue, diagrama técnico y paquetes. De acuerdo a los procesos de la división de distribución y recolección.

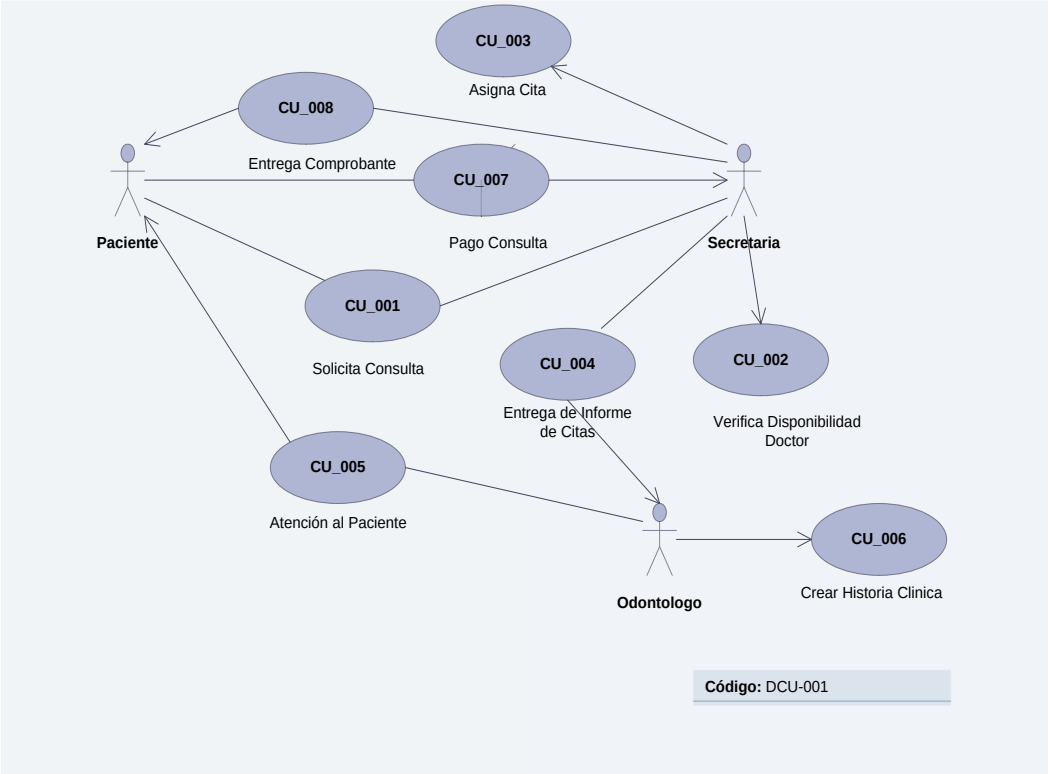

Figura 20. Diagrama de caso de uso de la clínica

3.5.5. Perfiles de usuario

- Paciente: solicita consultas y realiza pago de atención o servicios.
- Recepcionista: realiza atención a los pacientes, registra datos del paciente y asigna una cita.
- Odontólogo: atención de los pacientes y crea los historiales clínicos de los pacientes.

3.5.6. Documentación de casos de uso

A continuación en las figuras 21 y 22 se hace la descripción de los casos de usos identificados en el diagrama de negocios.

DCU-001 :	**Atención de consultas al paciente – Solicitud Verbal**
DOCUMENTACIÓN DEL DIAGRAMA DE CASO DE USO : DCU-001	
Código:	**CU-001**
Nombre:	Solicita consulta
Autor:	
Fecha:	04/03/2018
Descripción: El cliente solicita a la secretaria una consulta con el odontólogo	
Actores: Solicitante o paciente. Secretaria.	
Pre condiciones: Se debe tener un motivo de atención.	
Flujo Normal: 1.- El cliente consulta a la secretaria una cita 2.- La secretaria registra datos del paciente. 3.- La secretaria verifica la disponibilidad del doctor 4.- la secretaria comprueba los horarios 5.- La secretaria entrega datos de la cita	
Flujo alternativo:	
Post condiciones:	

Figura 21. Documentación del caso de uso, atención de consultas

DOCUMENTACIÓN DEL DIAGRAMA DE CASO DE USO : DCU-001	
Código:	**CU-008**
Nombre:	Entrega comprobante
Autor:	
Fecha:	04/03/2018
Descripción: El siguiente caso de uso consiste en la entrega y llenado del comprobante de pago por la consulta.	
Actores: Paciente Secretaria	
Pre condiciones: Haber realizado el pago de la consulta	
Flujo normal: 1.- La secretaria comprobará la realización del pago por la consulta. 2.- Ingresa datos del paciente al comprobante 3.- Ingresa datos del doctor que realizo la consulta 4.- Ingresa motivo de la consulta y costo del servicio. 5.- Imprime comprobante 6.- Entrega comprobante de pago al paciente.	
Flujo alternativo:	
Post condiciones:	

Figura 22. Documentación del caso de uso, entrega comprobante

3.5.7. Diagramas según UML

3.5.7.1. Diagramas de estructura.

a. Diagramas de clases.

En la figura 23 se identifica las clases, los atributos y los métodos que serán necesarios para elaborar el software. Como es que interactúan entre sí.

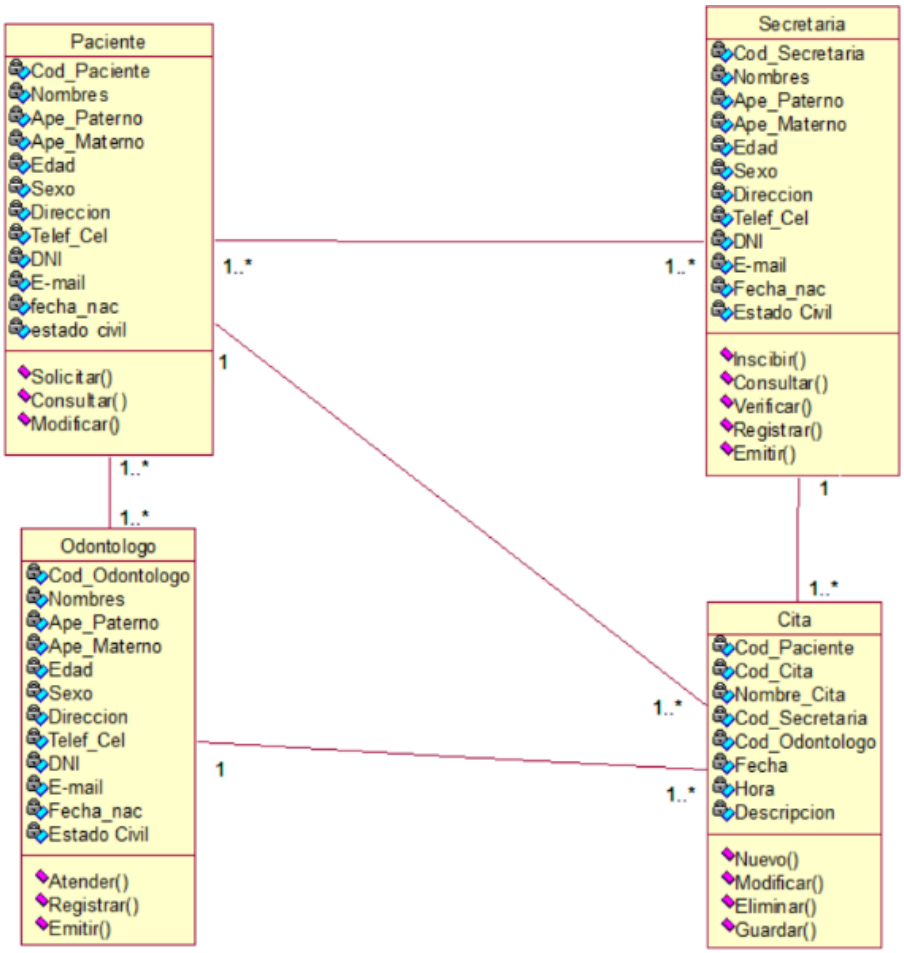

Figura 23. Diagrama de Clases

b. Diagramas de Objetos.

En la figura 24 se muestran los objetos.

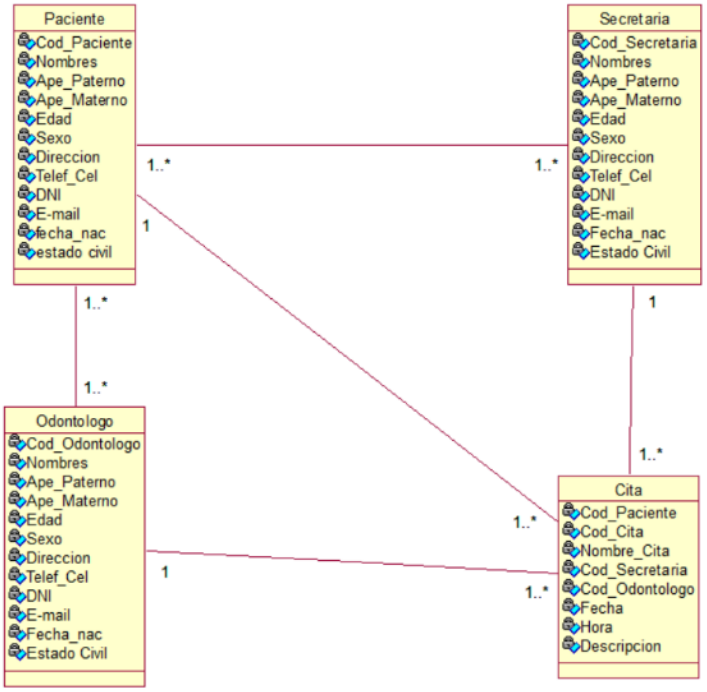

Figura 24. Diagrama de Objeto

c. Diagrama de Paquetes.

En la figura 25 se muestra cómo funcionará el sistema

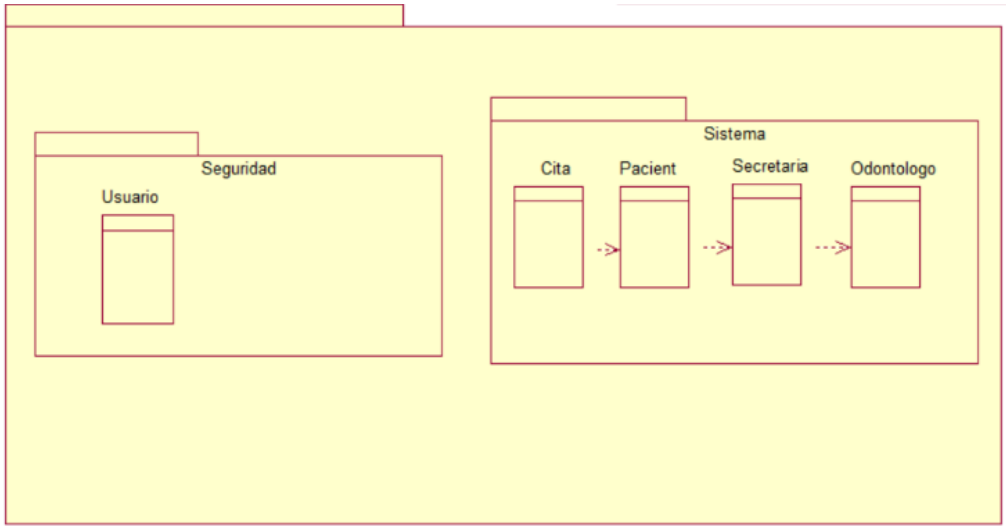

Figura 25. Diagrama de Paquetes

d. Diagrama de Componentes.

En la figura 26 se muestra como el sistema de software estará dividido en componentes y muestra la dependencia entre los componentes.

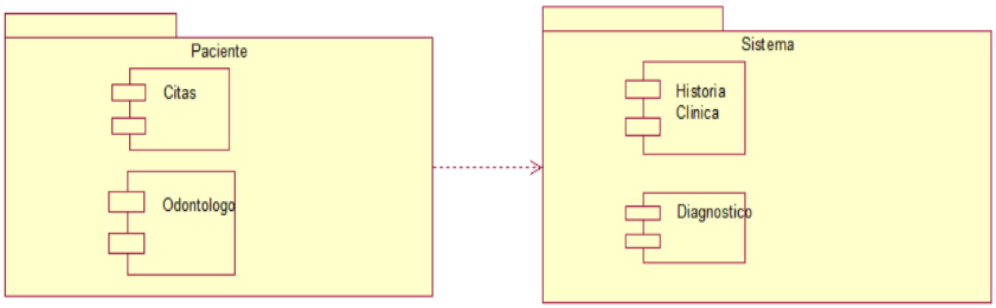

Figura 26. Diagrama de Componentes

3.5.7.2. Diagramas de comportamiento.

a. Diagrama de Casos de Uso.

En la figura 27 se muestra el comportamiento del sistema, la interacción entre los actores y se define los casos de uso.

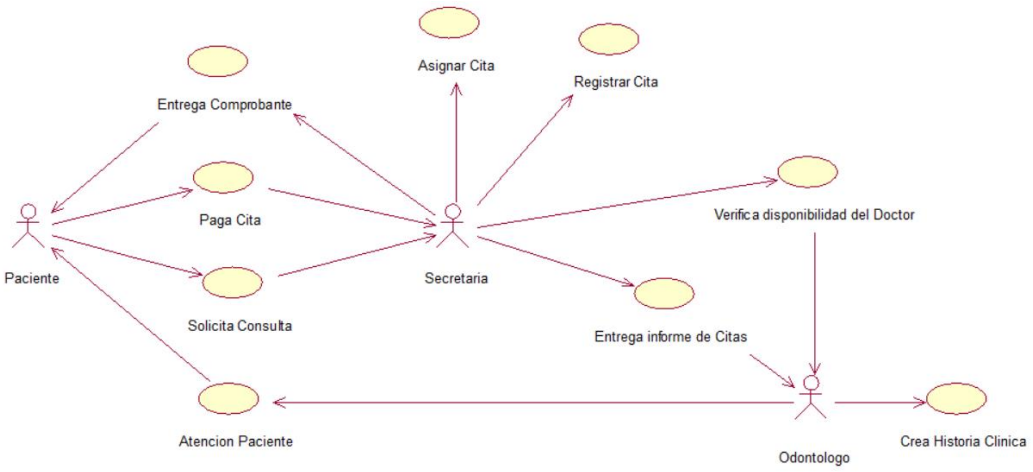

Figura 27. Diagrama de Casos de Usos

b. Diagrama de Estado.

En la figura 28 se muestra la secuencia de estados que pasa el objeto durante su vida.

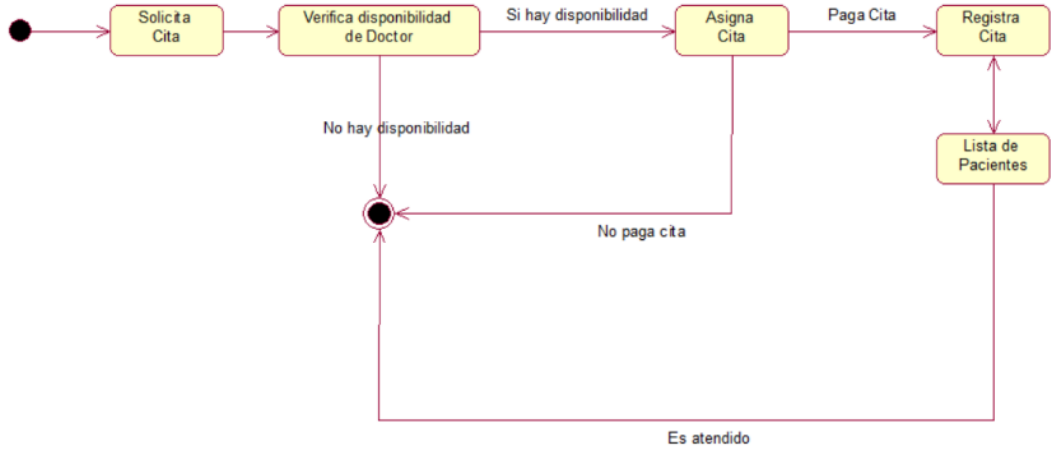

Figura 28. Diagrama de Estado

c. Diagrama de Actividad.

En la figura 29 se muestra el proceso del software como un flujo de trabajo, definiendo las acciones.

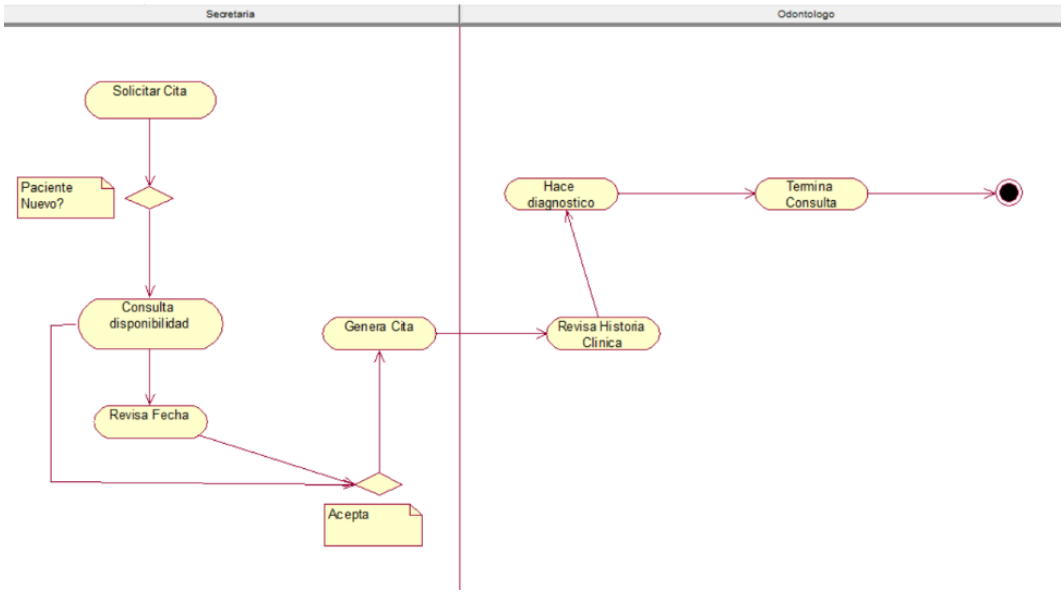

Figura 29. Diagrama de Actividad

d. Diagrama de Secuencia.

En la figura 30 se muestra el modelamiento de la interacción entre objetos en el sistema.

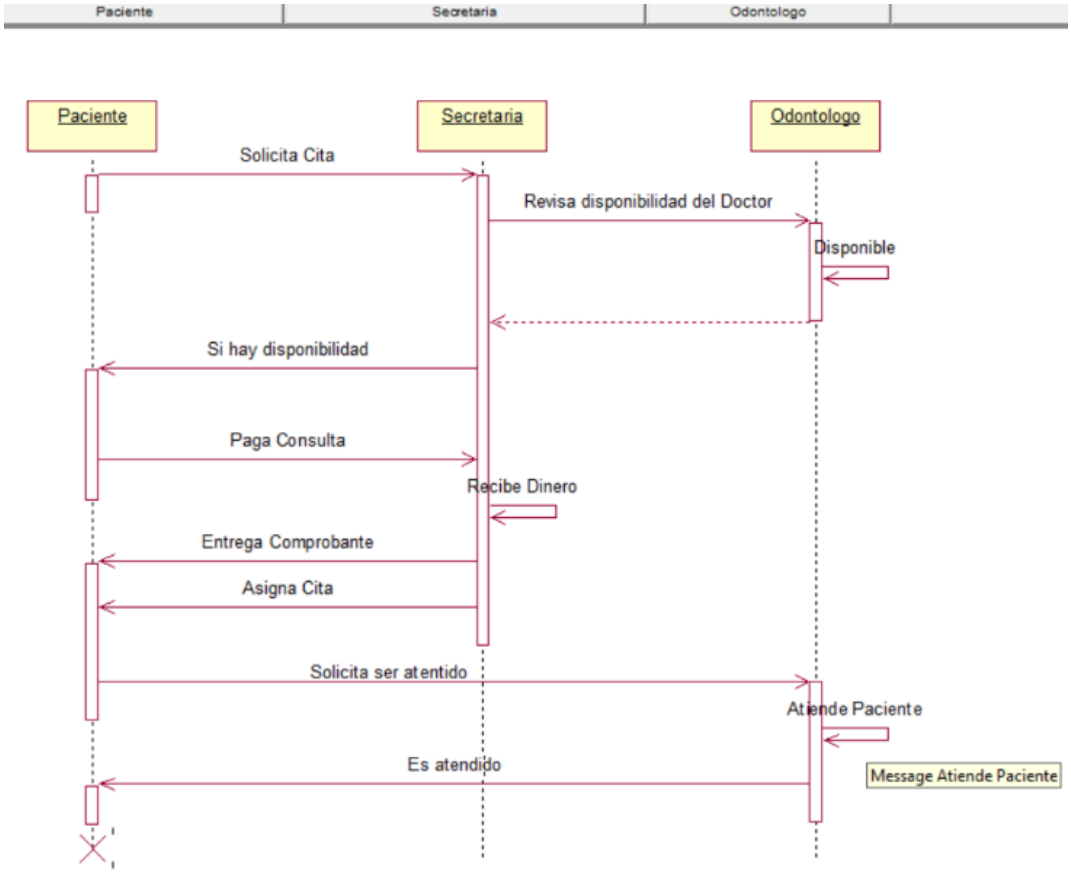

Figura 30. Diagrama de Secuencia

3.5.8. Distribución de paquetes de la solución del sistema

En la solución del sistema de atenciones de la clínica se ha realizado en netbeens 8.2, en la figura 31 podemos observar el nombre del proyecto se llama Sis_Clinica y está distribuido en paquetes siguientes:

- Datos: En este paquete va la conexión y los accesos a datos.
- Entidades: En este paquete se desarrolla las entidades en base a las clases del sistema.

- Interfaces: En este paquete se desarrolla las interfaces para poder consumir desde la capa lógica.
- Lógica: En este paquete se encuentran toda la lógica de negocios del sistema.
- Vistas: En el siguiente paquete se encuentran todos los formularios.

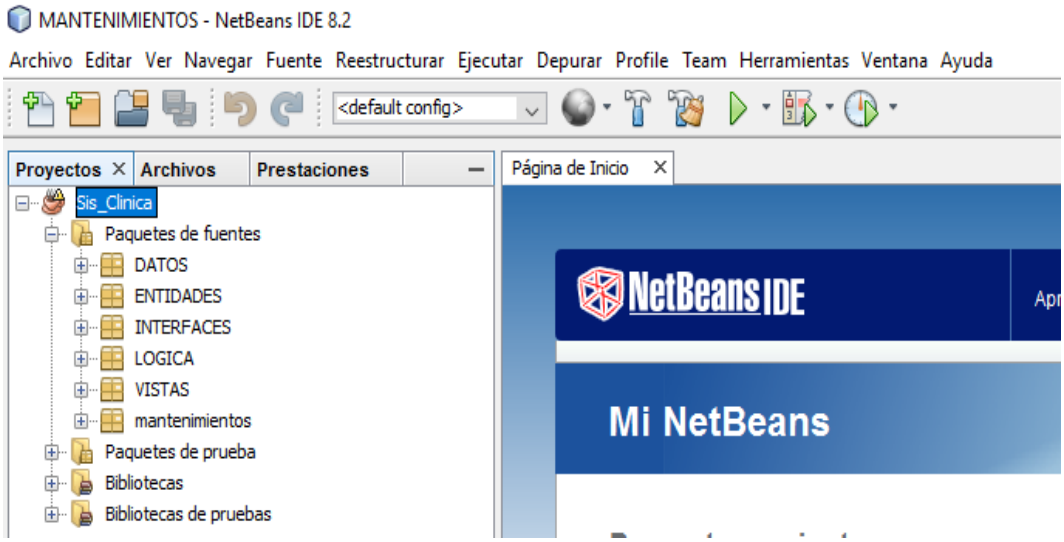

Figura 31. Esquema del proyecto de sistema de clínica

3.5.9. Código de la clase conexión

En la figura 32 se puede apreciar el código que se utilizará para establecer la conexión.

```
public class CONEXION {
  private Connection conection = null;

   public Connection getConection() {
      //Connection Conection=null;
      if (conection == null) {
         try {

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            String connectionUrl = "jdbc:sqlserver://Server:1433;" +

"databaseName=DbSistemaClinica;user=sa;password=123456789;";
            conection = DriverManager.getConnection(connectionUrl);
         } catch (SQLException e) {
            System.out.println("SQL Exception: "+ e.toString());
         } catch (ClassNotFoundException cE) {
            System.out.println("Class Not Found Exception: "+
cE.toString());
         }
      }
   return conection;   }   }
```

Figura 32. Código de conexión

3.5.10. Interfaz gráfica

3.5.10.1. Entrada al sistema.

En la figura 33 se muestra la interfaz de la entrada al sistema, los datos que se le solicitara al usuario.

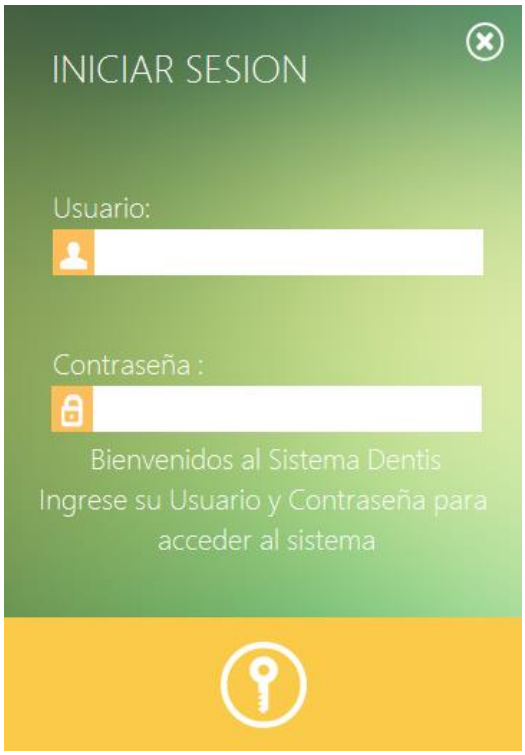

Figura 33. Formulario login

3.5.10.2. Programa principal del sistema.

En la figura 34 se muestra las opciones con las que contará el sistema.

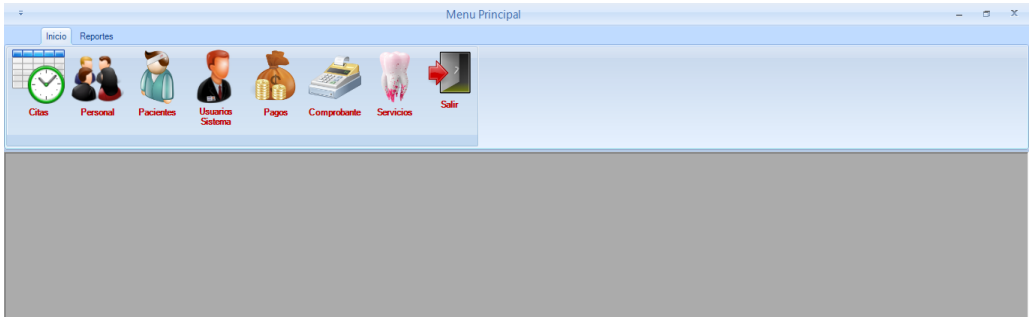

Figura 34. Formulario principal de la aplicación

3.5.10.3. Formulario de pacientes.

En la figura 35 se muestra el formulario de pacientes, todos los datos que contendrá.

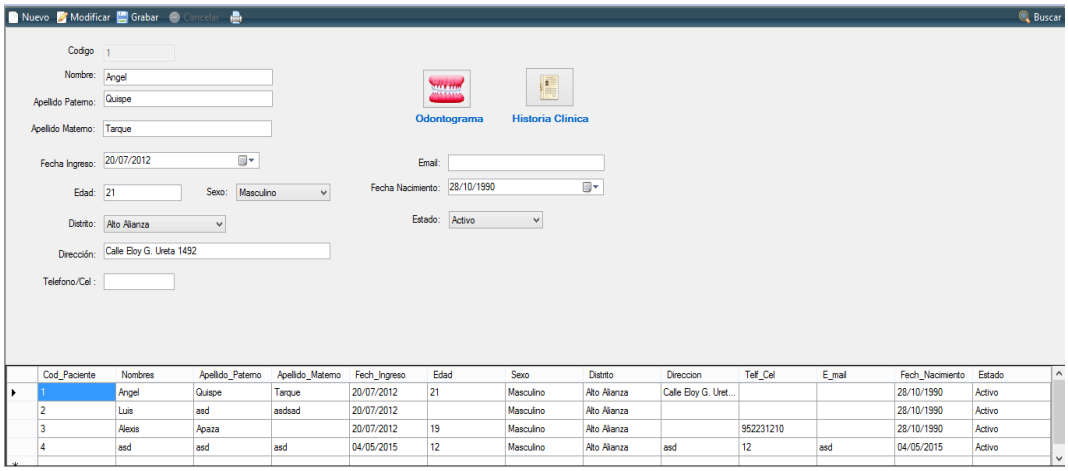

Figura 35. Formulario de pacientes

3.5.10.4. Formulario de servicios.

En la figura 36 se muestra el formulario de servicios, con las opciones que contará.

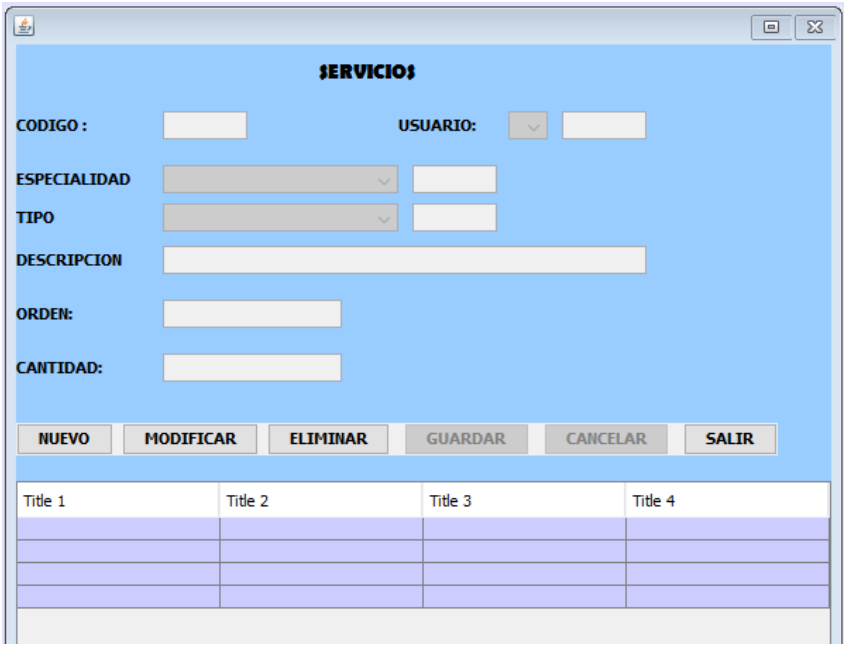

Figura 36. Formulario de servicios

3.5.11. Código fuente de Java

En la figura 37 se mostrará el código que se utilizó para el sistema.

```
public class DBPaciente implements InterfaceDaoPaciente {
    private Connection conexion = new CONEXION().getConection();


        public void agregarPaciente(Paciente paciente) {
            try{
                CallableStatement sql = conexion.prepareCall("{ call UPS_I_Paciente(?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?) }");
                sql.setInt("Id_Paciente", Paciente.getId_Paciente());
                sql.setInt("Id_TipoPaciente", Paciente.getId_TipoPaciente());
                sql.setString("Nombre", Paciente.getNombre());
                sql.setString("apellido", Paciente.apellido());
                sql.setString("Dni", Paciente.getDni());
                sql.setString("Direccion", Paciente.getDireccion());
                sql.setString("Fax01", Paciente.getFax01());
                sql.setString("Fax02", Paciente.getFax02());
                sql.setString("Correo_Electronico01", Paciente.getCorreo_Electronico01());
                sql.executeUpdate();

            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
```

Figura 37. Código fuente de Java

3.6. Representación de resultados

3.6.1. Descripción del trabajo de campo

Se ha realizado el sistema de atenciones de la clínica y se ha realizado una simulación en la clínica con un éxito del 90 % en fase prueba. Los involucrados ya poseen conocimientos en la utilización de manejo del sistema, ya que el software está diseñado a medida y es amigable respecto del manejo.

3.6.2. Análisis de la encuesta

En esta parte del proyecto se explicará las encuestas que se han aplicado a los empleados de las diferentes áreas y la opinión con respecto a la implantación en fase de prueba del sistema de atenciones de pacientes, a su vez medir el nivel de capacitación en computación del personal.

3.6.2.1. Tabulación por pregunta, opción y porcentaje (%) por opción.

a. Aplicación de este sistema informático.

¿Cree usted que se debería automatizar el proceso de Atención de pacientes para tener un mejor control?, en la tabla 4 y la figura 38 veremos cómo se distribuyen las respuestas a la pregunta realizada.

Tabla 4
Resultado de la aplicación del sistema informático

Respuesta	Cantidad	Porcentaje (%)
Sí	4	80
No	1	20
Total	5	100

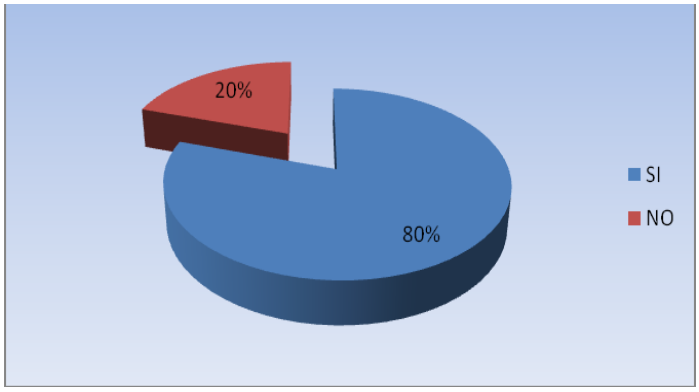

Figura 38. Resultados de la aplicación del sistema informático

- *Análisis.*

La pregunta fue hecha con el objetivo de saber su opinión acerca de la implementación del sistema de atenciones de pacientes. El 80 % de los empleados del área donde se implantará están de acuerdo con el sistema. Y solo un 20 % tiene aspectos contradictorios con el sistema.

b. Importancia del sistema informático hoy en día.

¿Considera usted que la aplicación del Sistema proporcionará mejoras en su plataforma de trabajo? (solo para el área de informática), en la tabla 5 y la figura 39 veremos cómo se distribuyen las respuestas a la pregunta realizada.

Tabla 5
Resultado de la importancia del sistema informático

Respuesta	Cantidad	Porcentaje (%)
Sí	4	100
No	0	0
Total	4	100

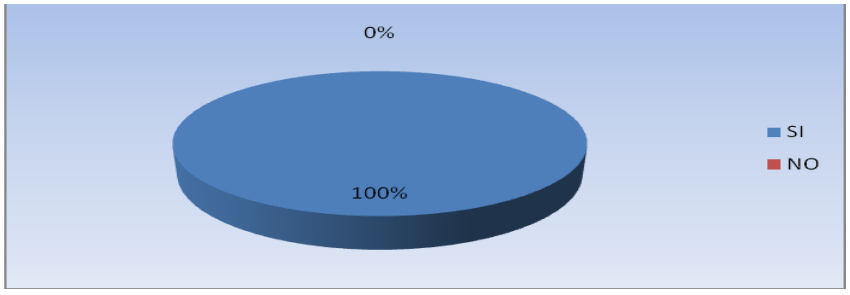

Figura 39. Resultados de la importancia del sistema informático en porcentaje

- *Análisis.*

El 100 % de los trabajadores del área de informática respondieron que si porque consideran que la aplicación del sistema ayudara a brindar más servicios y el código sea más flexible.

c. El sistema de atención de pacientes mejorará nuestras labores.

¿Cree que el desarrollo del sistema será eficaz en la elaboración de reportes para la toma de decisiones en la gerencia de administración? , en la tabla 6 y la figura 40 veremos cómo se distribuyen las respuestas a la pregunta realizada.

Tabla 6

Resultado de la pregunta el sistema de atención de pacientes

Respuesta	Cantidad	Porcentaje (%)
Sí	45	90
No	5	10
Total	50	100

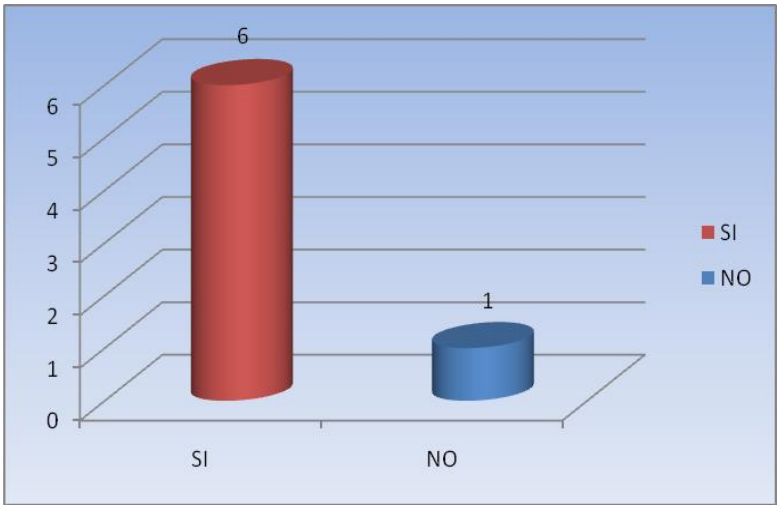

Figura 40. Resultados de si el sistema mejorará la atención.

- *Análisis.*

El 90 % de los empleados de la administración, creen que el sistema ayudará en la elaboración de los reportes al tener la información de manera inmediata.

3.6.3. Discusiones

- Los resultados luego de implantar el sistema denominado sistema de atención de pacientes de la clínica Ebenezer", se discutió los resultados en función a los objetivos trazados.

- Se realizó la verificación proporcionada por los medios estadísticos, pero siendo una implementación de software y basada en la experiencia del área de informática de la clínica se ha visto eficacia del software y como brinda el debido control de las atenciones de los pacientes, asimismo le brinda una herramienta muy útil a la clínica.

- Por eso vemos por conclusión que el desarrollo del software ha cumplido con el control de las atenciones médicas.

- Asimismo se tiene como resultado un mejor control en las historias clínicas de los pacientes. Y un control adecuado para otorgar las citas brindando así un servicio adecuado a nuestros clientes.

- Con la implantación del sistema de atenciones de paciente las búsquedas de las historias clínicas se realizan rápidamente reduciendo el tiempo de búsqueda de dicha historia.

- En la siguiente tabla 7 podemos ver los tiempos de mejora en el proceso de búsqueda de historias clínicas.

Tabla 7

Tiempo de mejora en el proceso de búsqueda

Tareas	Proceso manual	Proceso automatizado
Generación de reportes	15 minutos	3 minutos
Reportes de fichas	15 minutos	5 minutos
Reportes estadísticos mensuales	3 horas	30 minutos

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

Primera. El lograr desarrollar el sistema de atención de pacientes y el control de ingresos manejando una base de datos en la clínica Ebenezer, permitió cumplir con el objetivo general, ya que se ha contribuido de manera eficiente minimizando el tiempo en el registro de citas, la actualización y mayor precisión de la información.

Segunda. Se logró identificar los procesos y actividades del sistema manual de atención de pacientes que contienen un flujo de información considerable.

Tercera. Se logró establecer el modelo físico y lógico para una mejor construcción de la base de datos.

Cuarta. Se logró diseñar el software utilizando como lenguaje de programación Java, cumpliendo así con uno de los objetivos propuestos, el cual era la elaboración de software de alta calidad que pueda satisfacer las necesidades de sus usuarios finales con un presupuesto y tiempo establecido.

Quinta. Se logró realizar la base de datos utilizando el sistema de gestión de base de datos SQL Server 2008, que permitió desarrollar el modelo físico de la base de datos, procedimientos almacenados y administración de usuarios.

4.2. Recomendaciones

Primera. Es conveniente realizar periódicamente un mantenimiento al software y ver si todavía se encuentra acorde con los procesos que se identificó inicialmente.

Segunda. Es conveniente realizar periódicamente una revisión de los procesos y las actividades que se llevan a cabo en la clínica para poder ver si se pueden rediseñar o eliminar algunos de ellos para un mejor servicio, esto implicaría que se vuelva a utilizar la Metodología RUP para una mejor visión de los procesos que se dan.

Tercera. Para obtener un mejor control de variables que pueda ayudarnos en el desarrollo de aplicaciones manteniendo una ordenada implementación de éstas, es necesario seguir metodologías y estándares que permitan estar vigentes en un mundo competitivo.

Cuarta. Es necesario revisar el software para la actualización de librerías en Java, para su mejor funcionamiento y estar acorde con los avances tecnológicos.

Quinta. Se recomienda hacer una revisión periódica de la base de datos implementada en SQL Server, con el fin que se esté almacenando de manera correcta la información de la clínica.

REFERENCIAS BIBLIOGRÁFICAS

Cairo, O. y Guardati, S. (2006). *Estructura de datos*. México: Mcgraw-Hill.

Ceballos, F. (2010). *JAVA 2 curso de programación*. España: Alfaomega.

Eckel, B. (2005). *Pensando en JAVA*. México: Prentice Hall.

Fowler, M. y Scott, K. (1999). *UML gota a gota*. México: Prentice-Hall Hispanoamericana. Recuperado de https://ingenieriasoftware2011. files.wordpress.com/2011/07/uml-gota-a-gota.pdf

Holzner, S. (2005). *La Biblia del JAVA 2*. Madrid: Anaya Multimedia.

Jacobson, I. y Booch, G. y Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software*. Madrid: Pearson Educación.

Lazo, W. (1995). *Programación estructurada en C*. Perú: Universidad Antenor Orrego de Trujillo.

Liza, C. (2016). *Algoritmos y su codificación C++*. Perú: Grupo Creadores.

Morgan, L. (1999). *Descubre JAVA 1.2*. México: Prentice Hall.Mike M.

Schmuller, J. (1999). *Aprendiendo UML en 24 horas*. México: Prentice-Hall Hispanoamericana. Recuperado de https://www.u-cursos.cl /ingenieria/2008/1/CC51H /1/material_docente/aprendiendo UML en 24 horas

Vargas, M. (2013). *Administrando base de datos con SQL server 2012*. Lima: Ritisa.