Universidad de las Ciencias Informáticas

Facultad 3

CEIGE

Desarrollo del Componente Ingresos para la variante de escritorio del Sistema Integral de Seguros Nacionales

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autora:

Kenia Aparicio Tartabull

Tutores:

Ing. Lisandra Tamayo Espinosa

Ing. Luis Ángel Castillo Rodríguez

Ing. José Daniel Sánchez Hechavarría

La Habana, junio del 2017

"Las nuevas tecnologías de las comunicaciones han dividido al mundo entre los conectados y los no conectados a las redes globales."

Fidel Castro Ruz

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____
Kenia Aparicio Tartabull
Firma del Autor

_____	_____	_____
Ing. Lisandra Tamayo Espinosa	Ing. Luis Ángel Castillo Rodríguez	Ing. José Daniel Sánchez Hechavarría
Firma del Tutor	Firma del Tutor	Firma del Tutor

Agradecimientos

Primeramente, le agradezco a mi mamá por estar siempre a mi lado, por guiarme en todo momento, por sus consejos y su apoyo incondicional y por ser la mujer, la abuela, la amiga y la mejor madre del mundo y del universo.

Muchísimas gracias a mi padre, a mi hermano y a mi familia en general por estar de una forma u otra siempre presentes en mí. Los quiero mucho.

Le agradezco a Michael por apoyarme y soportarme siempre, por estar junto a mí en las buenas y malas.

A mis compañeras de batallas y gozaderas Yarilis, Dianelis, Asiledi, Esmirna, Célida, las Claudias, Dalilis yYanet. También a los compañeros del aula Ernesto Carlos, Yoe, y los demás compañeros que conocí desde primer año que, aunque no estén algunos se les quiere mucho.

Agradecida con mis tutores, el tribunal y los profesores que durante cinco años me apoyaron para esforzarme cada día y poder obtener este mayor premio que es graduarme de Ingeniera Informática.

DEDICATORIA

Este trabajo va dedicado a mi familia especialmente a mi madre y a mi abuelita Fortuna, por ser los ejemplos a seguir durante toda mi vida. Las amo mucho.

RESUMEN

Actualmente, en Cuba se realiza un gran esfuerzo en aras de mejorar los resultados económicos nacionales, para lo cual se hace uso de las Tecnologías de la Información y las Comunicaciones, en favor de lograr una mayor eficiencia y control sobre los recursos. La Empresa de Seguros Nacionales necesita un sistema que garantice registrar los ingresos cuando exista interrupciones en el servidor. Con este trabajo se propone desarrollar un componente que permita gestionar los ingresos, las comisiones adicionales y los ajustes de comisión de los agentes de la Empresa de Seguros Nacionales de Cuba, para el caso que la estación central esté temporalmente sin servicio; tomando como guía para ello, la metodología de desarrollo utilizada en la Universidad de las Ciencias Informáticas, acorde a las peculiaridades de la entidad y las tendencias actuales del desarrollo de software. A lo largo de tres capítulos se describe el diseño, implementación y validación del Componente Ingresos, lo cual permitió a partir de los requisitos previamente identificados, desarrollar una aplicación de escritorio para la gestión de los ingresos en menor tiempo, adaptable a la estructura organizativa de la ESEN.

Palabras claves: *ESEN, ingresos, aplicación de escritorio.*

SUMMARY

Currently, a great effort is being made in Cuba in order to improve national economic performance, using Information and Communication Technologies, in order to achieve greater efficiency and control over resources. The National Insurance Company needs a system that guarantees to record the income when there are interruptions in the server. This paper proposes to develop a component to manage the revenues, additional commissions and commission adjustments of the agents of the National Insurance Company of Cuba, in case the central station is temporarily without service; taking as a guide for this, the development methodology used in the University of Computer Science, according to the peculiarities of the entity and current trends in software development. Throughout three chapters the design, implementation and validation of the Revenue Component was described, which allowed, from the previously identified requirements, to develop a desktop application for income management in a shorter time, adaptable to the organizational structure of The ESEN.

Keywords*: ESEN, revenues, desktop application.*

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

El desarrollo informático ha disparado el uso masivo de Internet y de los medios sociales, produciendo un cambio en el modelo de negocio de las empresas. Los clientes pasan de ser solo consumidores a creadores del producto que van a consumir. De este modo es necesario crear estrategias que se basen en nuevas tecnologías, para brindar soluciones a las empresas, entidades o beneficiarios de esta, que sean consistentes o estén adecuadamente integradas a sus estrategias generales del negocio (Sayas, 2013). Debido a esto muchas empresas tienen informatizados sus procesos claves que les proveen ventajas sobre la competencia, mayores ingresos, rapidez en el procesamiento de datos, control de ventas y atención al cliente.

En la actualidad las empresas destinan sus esfuerzos en la disponibilidad de sus servicios y no en crear estrategias que permitan que estos servicios funcionen mientras la red está temporalmente caída, ocasionando como desventaja, que la comunicación constante con el servidor que ejecuta una aplicación, establece dependencia con una buena conexión a Internet. Además, el servidor debe tener las prestaciones necesarias para ejecutar la aplicación de manera fluida, no sólo para un usuario sino para todos los que la utilicen de forma concurrente. Por concepto de las interrupciones que pueda ocasionar la caída de conexión, las empresas pierden alrededor de 8000 dólares por cada minuto que sus servicios no están online (Villarrubia, 2013).

El avance de las nuevas Tecnologías de la Información y las Comunicaciones (TIC), también ha provocado profundos cambios en el contexto social y laboral en el que se desenvuelve el hombre moderno. Como resultado directo de este proceso ha emergido el concepto de Sociedad de la Información, el cual ha irrumpido como un nuevo fenómeno a escala mundial donde la información y el conocimiento ocupan un lugar privilegiado en la sociedad (Naranjo, 2012).

La alta disponibilidad y accesibilidad de la información en la sociedad actual ha abierto el espectro a un cúmulo de aplicaciones orientadas a diferentes sectores como por ejemplo: la medicina, la administración pública y empresarial, transporte público y la educación. Estos sistemas responden a las necesidades específicas del dominio o negocio para el cual fueron creados, motivo por el cual, el grado de descentralización y dispersión de los datos almacenados, representa un tema en consideración, especialmente en una sociedad que consume y crea enormes volúmenes de información.

La penetración de las tecnologías en casi todas las áreas de la sociedad obliga a utilizar varias soluciones informáticas en función de sus intereses y objetivos, para lo cual se debe dedicar un tiempo finito y asimilar la información contenida. Generalmente, estas aplicaciones podrían estar integradas, y en ese caso, dichas personas estarían ahorrando tiempo útil.

Cuba, a pesar de las dificultades y de los obstáculos que provoca el bloqueo económico, no se encuentra ajena a los avances tecnológicos y está enfrascada en la informatización de la sociedad. Es por ello que la Empresa de Seguros Nacionales de Cuba (ESEN) también está guiada a informatizar sus procesos. Dicha empresa, tiene como objeto empresarial desarrollar operaciones de seguros y reaseguros en moneda nacional y divisa, así como realizar actividades preparatorias y complementarias al seguro, dirigidas a la evaluación de riesgos y prevención de daños. También le compete ofrecer servicios de inspección, tasación y ajustes de averías, cálculos actuariales y prevención del riesgo en bienes asegurados en ambas monedas. Esta entidad cuenta con una Oficina Central, varias direcciones provinciales de seguro (Unidades Empresariales de Base (UEB)) en todas las provincias del país incluyendo el municipio especial Isla de la Juventud y negocios especiales en seguros a personas jurídicas radicadas en la Capital con representaciones por territorios en cada provincia y con una red de agentes (tanto personas naturales, jurídicas, cubanas o extranjeras radicadas en Cuba) con la intención de acercar sus servicios al cliente y así poder brindarle una atención personalizada y con calidad (ESEN, 2016).

Actualmente, la ESEN trabaja a partir de dos sistemas informáticos (INFOSEG y SIGES) que permiten la gestión de su información, dichos sistemas se ubican en las propias UEB del país, por lo que los informáticos tienen acceso a su código fuente y base de datos, permitiéndoles alterar su programación y la información almacenada, poniendo en riesgo su integridad. Existe falta de uniformidad en la información, lo cual limita conocer de forma rápida y fácil, por parte de la Oficina Central, los datos necesarios para la toma de decisiones, el seguimiento y el control de sus procesos. En dichos sistemas, las bases de datos se encuentran descentralizadas y son distintas para cada uno de ellos, lo que posibilita que una persona se asegure en el mismo producto y con las mismas condiciones en dos provincias diferentes, lo cual constituye una violación a los procedimientos establecidos. Por otro lado, en ambas aplicaciones se cuenta como dos clientes distintos, a los asegurados que tienen dos contratos de seguro, cuando realmente es un asegurado con dos pólizas. Además, un trabajador de la empresa, que jurídicamente no tiene autorización para registrar los ingresos, solamente introduce la información del cliente en dichas aplicaciones, permitiendo convertir dicho ingreso en un registro contable, el cual es procesado y registrado solo por contabilidad. Todo esto obliga a realizar la gestión manual y conlleva a la pérdida de tiempos valiosos para la toma de decisiones, así como el riesgo de errores humanos en los datos ofrecidos.

Teniendo en cuenta la importancia de los procesos de la ESEN y la necesidad de resolver los problemas existentes en la gestión de los mismos, en la Universidad de las Ciencias Informáticas, específicamente en el Centro de Informatización de la Gestión de Entidades (CEIGE), se desarrolla el Sistema Integral de Seguros Nacionales (SISEN) , con el propósito de informatizar los procesos que se llevan a cabo en el área comercial de la ESEN y desplegar dicho sistema en una estación central a la cual se conectarán

todas las UEB del país. El despliegue de esta aplicación en toda la red nacional de la ESEN obligaría a desechar los antiguos métodos, permitiendo ahorrar tiempo, recursos y obtener rapidez en cuanto a la atención al cliente.

El SISEN es un sistema web compuesto por 21 módulos, de los cuales Concertación e Ingresos influyen en los resultados económicos de la empresa. La concertación permite comercializar las pólizas de seguro de la ESEN, sin embargo, para que finalice correctamente dicho proceso se debe registrar los ingresos monetarios obtenidos durante la gestión de las pólizas, por lo que es vital priorizar la gestión de los ingresos. Para lograr la gestión de dicho proceso es necesario que las unidades estén conectadas a la red, y debido a que las conexiones a la red están expuestas a fallar por interrupciones en el servicio eléctrico, en el sistema de monitoreo, errores de configuración, sobrecarga del sistema o explotación de las vulnerabilidades del servidor, el equipo de trabajo de la ESEN propone el desarrollo de una solución que permita continuar trabajando mientras el servidor esté fuera de servicio. En caso de que las unidades estén desconectas significaría pérdidas Monetarias para la empresa. Principalmente por la pérdida de tiempo, que implicaría insatisfacción y desconfianza en los clientes, poniendo en riesgo la credibilidad de la empresa y la oportunidad de tener clientes futuros. La disminución de las ventas es otro factor que influye económicamente, pues si no se realizan ingresos, la empresa estancaría y llegaría hasta un punto donde los costos internos se elevan y la situación se volvería insostenible. Por otro lado, si la empresa decrementa su productividad, reduciría la posibilidad de aumentos de sueldos de los trabajadores, lo que provocaría pérdida del personal.

La problemática previamente descrita origina la necesidad de dar respuesta al siguiente **problema a resolver**: ¿Cómo minimizar el tiempo de la gestión de los ingresos del SISEN mientras el servidor central está temporalmente sin servicio?

Tomándose como **objeto de estudio**: Proceso de gestión de almacenamiento de información fuera de línea, para centrar la investigación se tiene como **campo de acción**: El proceso de ingreso de información fuera de línea en el SISEN.

Para dar solución a la problemática descrita se establece como **objetivo general**: Desarrollar el componente Ingresos para la variante de escritorio del SISEN.

Se trazan como **objetivos específicos**:

- Elaborar el marco teórico-conceptual para fundamentar la investigación.
- Diseñar la solución de software basado en los requisitos previamente identificados.
- Implementar el componente Ingresos para la variante de escritorio del SISEN.
- Validar el correcto funcionamiento del sistema mediante las pruebas.
- Validar la investigación para medir la eficiencia del componente basado en el tiempo.

La investigación está basada en la siguiente **idea a defender**:
Con el desarrollo del componente Ingresos para la variante de escritorio, se minimizará el tiempo en la gestión de los ingresos del SISEN mientras el servidor central está temporalmente desconectado.

Para el desarrollo de la investigación se hizo uso de métodos científicos de investigación siendo estos la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. Se clasifican en teóricos y empíricos, los cuales están dialécticamente relacionados (González, 2012). En la investigación se utilizaron los siguientes **métodos teóricos:**

Analítico-Sintético: Se utilizó con el objetivo de analizar los documentos, teorías y otros materiales que permitieron la comprensión de los elementos relacionados con el objeto de estudio.

Histórico-Lógico: Se empleó en el estudio de cómo ha evolucionado el tema que se está investigando, de las diferentes herramientas existentes, sus características, ventajas y desventajas.

Modelación: Este método se utiliza principalmente en la investigación debido a que permite la representación de diferentes ideas de forma gráfica, lo que posibilita un mayor entendimiento en la etapa de implementación del sistema. También se ha utilizado para modelar todos los diagramas correspondientes a la etapa de análisis, diseño e implementación del módulo a desarrollar en la presente investigación.

Métodos empíricos:
Observación: Se utiliza para observar cómo funciona actualmente el sistema (SISEN) que desarrolla el equipo de proyecto de CEIGE.

Entrevista de tipo abierta: Se utiliza este método para el entendimiento del negocio y los requerimientos que el sistema deberá satisfacer, además del proceso de familiarización con el personal capacitado sobre el tema a tratar, que en este caso es el funcionamiento del módulo Ingresos. Se emplea en la validación de la investigación para medir el tiempo que demora el proceso de gestión de la información en el componente.

El trabajo de diploma está definido estructuralmente en tres capítulos, descritos a continuación:

Capítulo 1: Fundamentación teórica: En este capítulo se describe el estado del arte de la investigación, se realiza el análisis de los conceptos fundamentales relacionados con el tema en cuestión y se fundamenta sobre las técnicas, artefactos de diseño y herramientas utilizadas.

Capítulo 2: Análisis y diseño del sistema: En este capítulo se hace un análisis y se describe la

propuesta de desarrollo del sistema, así como los requisitos funcionales y no funcionales previamente identificados por el cliente. Se describen el modelo de datos, el diagrama de clases del diseño, la arquitectura y patrones utilizados para el diseño del sistema. También se valida el diseño a partir de las métricas y se valoran los resultados obtenidos.

Capítulo 3: Implementación y prueba: En este capítulo se describe la construcción de la solución, explicando los aspectos principales de la implementación. Se realiza una descripción de las clases y funcionalidades del sistema. Además, se hace el diseño y se ejecutan las pruebas sobre dicha aplicación en busca de errores relacionados con su funcionalidad y se muestran los resultados obtenidos en la validación del sistema.

Capítulo 1: Fundamentación teórica.

1.1 Introducción

En el presente capítulo se abordan los principales conceptos asociados al dominio del problema, profundizándose en la gestión y técnicas de almacenamiento de información. Se fundamenta la selección de la metodología de desarrollo de software, las tecnologías y las herramientas a utilizar.

1.2 Conceptos básicos relacionados con el dominio del problema

Fuera de línea

- En telecomunicaciones: fuera de línea se utiliza para designar a una computadora o a un sistema que, está apagado o no accede a la red. (Leandro Alegsa, 2016)
- Estado de un usuario cuando no está conectado a un servicio de Internet. (Leandro Alegsa, 2016)
- Un equipo está fuera de línea cuando este está apagado. (Leandro Alegsa, 2016)
- Navegación sin conexión: Cuando es posible acceder a uno o más sitios web porque ya ha sido guardada una versión copiada localmente ya sea por el navegador web o por algún programa especial para tal fin. (Leandro Alegsa, 2016)
- Un contenido o aplicación está disponible fuera de línea, cuando no es necesario conectarse a una red para acceder al mismo o cuando existe una copia funcional utilizable localmente. (Leandro Alegsa, 2016)

A partir de lo expuesto anteriormente, se utiliza el último concepto para el entendimiento del término fuera de línea durante el desarrollo del trabajo.
A continuación, se exponen los principales conceptos del negocio:

Ingreso es el importe que se ingresa a la entidad por varios conceptos. (Cobas, 2015)

La **prima** es el pago que se le exige al asegurado, según sea la suma asegurada y tiempo de contrato establecido. (Castro, 2009)

La **póliza** es el documento en el que se hacen constar las condiciones del contrato de seguro y en la que se establecen los derechos y obligaciones de las personas que intervienen en él. (Castro, 2009)

Asegurado es la persona natural o jurídica titular del interés asegurado y por consiguiente, aquella cuyos bienes, persona y responsabilidades están expuestos al riesgo y que ejerce los derechos y responde por las obligaciones de la relación contractual[1] constituida. (Castro, 2009)

[1] **Contractual:** procedente del contrato o derivado de él.

Agente de seguros: Toda persona natural o jurídica que sea expresamente autorizada y vinculada a una entidad de seguros mediante un contrato de mandato oneroso[2] (agencia), se dedique de forma habitual y permanente a servir de mediadora entre esta y los posibles tomadores, conservando una cartera de seguros reconocida. (Castro, 2010)

El **Instrumento de cobro** es el mecanismo existente, proporcionado por el sistema financiero, para proceder al pago de los bienes o servicios adquiridos sin hacer uso del efectivo (billetes o monedas de curso legal). (Valdeande, 2012)

Váucher es el vale que da derecho a quien lo posee a adquirir determinados artículos o a disfrutar de un servicio. (Real Academia Española, 2014)

1.3 Gestión de almacenamiento de información

En la actual sociedad del conocimiento, las empresas y organizaciones generan, a un ritmo creciente, un altísimo volumen de datos. La información se ha convertido en uno de los ejes esenciales para todo tipo de negocios, tanto por su valor para conocer más certeramente a sus actuales y potenciales clientes, como para gestionar sus propios procesos de manera más eficiente. En tal sentido, surge la necesidad de familiarizarse con el concepto de gestión del ciclo de vida de la información e implementar el establecimiento de planes, normas y políticas de almacenamiento y protección de datos. (Solis, 2012) Para gestionar en forma eficiente la información es la correcta selección del hardware de almacenamiento en donde residirán los datos, así como una correcta política de seguridad. Es importante fijar un solo sitio de almacenamiento de documentos informáticos y que ese lugar tenga asignada una política de seguridad acorde. Si bien las políticas de almacenamiento dependerán de cada compañía, actualmente, la eficiencia e inteligencia en los equipos de almacenamiento y la rapidez con la que las compañías puedan tener acceso a su información crítica les brindará definitivamente una ventaja competitiva.

La elección de la plataforma de almacenamiento es clave para la obtención de esos beneficios, muchos sistemas tradicionales no cumplen con algunos requisitos básicos. La arquitectura correcta para almacenar los datos, debe adaptarse al cambio, es decir, debe transformarse de una plataforma rígida en un sistema dinámico, flexible y escalable, que además contribuya a cambiar desde una simple operación a la innovación de tecnología. Así mismo, una organización debe reconocer que no todos los datos pueden ser tratados y almacenados de la misma manera. La mejor manera de prepararse para ello es conocer cada tipo de información que se está almacenando, puesto que esto otorgará una visión más

[2] **Oneroso**: tipo de contrato en que ambas partes tienen obligaciones y ventajas económicas recíprocas.

clara para definir las necesidades, niveles de seguridad y la proyección de crecimiento del volumen de datos.

1.3.1 Técnicas de almacenamiento fuera de línea

Las aplicaciones desconectadas en línea tienen el costo de aumentar la complejidad, y se debería considerar si el soporte sin conexión está justificado. Existen situaciones en las que las tecnologías sin conexión son útiles para proteger las interrupciones del servidor o de la red.

Las tecnologías fuera de línea admiten el almacenamiento en caché y un control detallado del proceso de dicho almacenamiento. Por lo tanto, las aplicaciones pueden arrancar rápidamente y mostrar los datos al instante. Existen dos definiciones de dichas tecnologías, las cuales hacen ambiguo a este término. En un nivel, las "tecnologías sin conexión" podrían definirse como "tecnologías que soportan aplicaciones fuera de línea" o "tecnologías que operan fuera de la nube". Es evidente que existe coincidencia entre las dos definiciones; las aplicaciones sin conexión ciertamente necesitan operar fuera de la nube. (Mahemoff, 2013)

El almacenamiento fuera de línea se trata de capturar datos específicos generados por el usuario o recursos en los que el usuario ha expresado interés. A continuación, se ofrece una breve descripción de algunas de las técnicas de almacenamiento consultadas en la bibliografía (Gauchat, 2012) y (Lancker, 2013).

Web Storage (también llamada **Local Storage) -Almacenamiento web**

Es una forma conveniente de almacenamiento fuera de línea, siendo solo una simple estructura de pares clave-valor como un objeto de JavaScript, donde se puede almacenar para una clave un determinado valor. También existe una variante a localStorage: sessionStorage. Esta es usada por temas de seguridad, donde la información es eliminada cuando la ventana del navegador es cerrada.
El almacenamiento local es seguro, y se pueden almacenar localmente grandes cantidades de datos, sin afectar el rendimiento del sitio web. Esta técnica es por origen (por dominio y protocolo), lo que significa que todas las páginas, desde un origen, pueden almacenar y acceder a los mismos datos. Posee ausencia de transacciones.

Web SQL Database - (Base de datos Web SQL)

Es una base de datos desconectada donde las implementaciones de hoy en día están realizadas con SQLite, un motor SQL de código abierto de uso general. Igualmente, a una base de datos tradicional, tiene una estructura totalmente relacional, que permite consultar y manipular rápidamente datos a través de uniones de tablas. Además, tiene soporte para transacciones, su base de datos está protegida

del tipo de condiciones de competencia que pueden surgir con el almacenamiento web. Mediante Web SQL Database, la W3C[3] ofrece una API estándar destinada a manipular bases de datos en el lado del cliente mediante peticiones SQL de forma asíncrona.

Indexed DB - (Base datos indexado)

Esta técnica se encuentra entre las técnicas Web Storage y Web SQL Database. Es relativamente simple y veloz. Se basa en tomar un mapeo estándar, como el localStorage, pero indexando en ciertos campos dentro de la información almacenada. Como resultado, se puede obtener consultas rápidas y la posibilidad de utilizar transacciones. Indexed DB no es una base de datos propiamente dicha, por lo que no se tienen tablas, ni estructuras de datos al estilo SQL, lo que significa que es una serie de colecciones de objetos.

Filesystem API- (API de Sistema de archivos)

La interfaz de la API de entradas de archivos y directorios se utiliza para representar un sistema de archivos. Estos objetos pueden obtenerse de la propiedad filesystem en cualquier entrada del sistema de archivos. Posibilita el almacenamiento de contenido binario (como texto plano), creando carpetas y jerarquía de las mismas permitiendo de esta forma guardar grandes estructuras de datos. Con esta API, una aplicación web puede crear, leer, explorar y editar una sección de prueba del sistema de archivos local del usuario.

Se determina utilizar el funcionamiento de la técnica Web SQL Database, ya que se ajusta al caso que ocupa la presente investigación, donde es necesario almacenar información en una base datos local, con la diferencia de que es para una aplicación de escritorio. A pesar de que el resto de las técnicas estudiadas presentan gran utilidad en varios campos del almacenamiento de datos de forma desconectada, no se adecuan a las necesidades del presente trabajo.

1.4 Metodología de desarrollo

Metodología de desarrollo para la actividad productiva de la UCI **(Variación AUP-UCI):** Se basa en una variación de la metodología “Proceso Unificado Ágil” (AUP por sus siglas en inglés) en unión con el modelo CMMI-DEV versión 1.3.

La metodología está compuesta por tres fases principales: inicio, ejecución y cierre.

Inicio: En el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener

[3] **W3C:** (World Wide Web Consortium), es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo.

información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto. (Universidad de las Ciencias Informáticas, 2015)

Ejecución: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. (Universidad de las Ciencias Informáticas, 2015)

Cierre: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto. (Universidad de las Ciencias Informáticas, 2015)

Debido a que el proyecto tiene un ciclo de vida mayor a la investigación realizada, solo se transitará por la fase de Ejecución. En la misma se realizará el diagrama de clases del diseño, el diseño de la base de datos y los diseños de casos de prueba. Para ello se emplearán las disciplinas de análisis y diseño, implementación, pruebas internas y de aceptación, con el fin de obtener el componente Ingresos correctamente diseñado e implementado.

La metodología AUP en su variación UCI, presenta cuatro escenarios los cuales se desarrollan en la fase de Ejecución. Debido a que el presente trabajo parte de los requisitos identificados previamente, se emplea el escenario 3 que plantea como técnica de encapsulación de requisitos DRP (Descripción de requisitos por procesos) (Cobas, 2015).

Se utiliza AUP-UCI debido a que los proyectos productivos en la universidad se rigen por esta metodología para guiar el proceso de desarrollo del software.

1.5 Requisitos

La ingeniería de requerimientos es una de las acciones importantes de la ingeniería de software que comienza durante la actividad de comunicación y continúa en la de modelado. Debe adaptarse a las necesidades del proceso, del proyecto, del producto y de las personas que hacen el trabajo. Proporciona el mecanismo apropiado para entender lo que desea el cliente, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación y administrar los requerimientos a medida que se transforman en un sistema funcional (Pressman, 2010).

1.5.1 Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también

pueden declarar explícitamente lo que el sistema no debe hacer (Sommerville, 2005).

1.5.2 Requerimientos no funcionales

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. A menudo se aplican al sistema en su totalidad. Normalmente no se aplican a características o servicios individuales del sistema (Sommerville, 2005).

A continuación se muestra los atributos de calidad para los requerimientos no funcionales según la ISO 9126, extraídos de (Cobas, 2015).

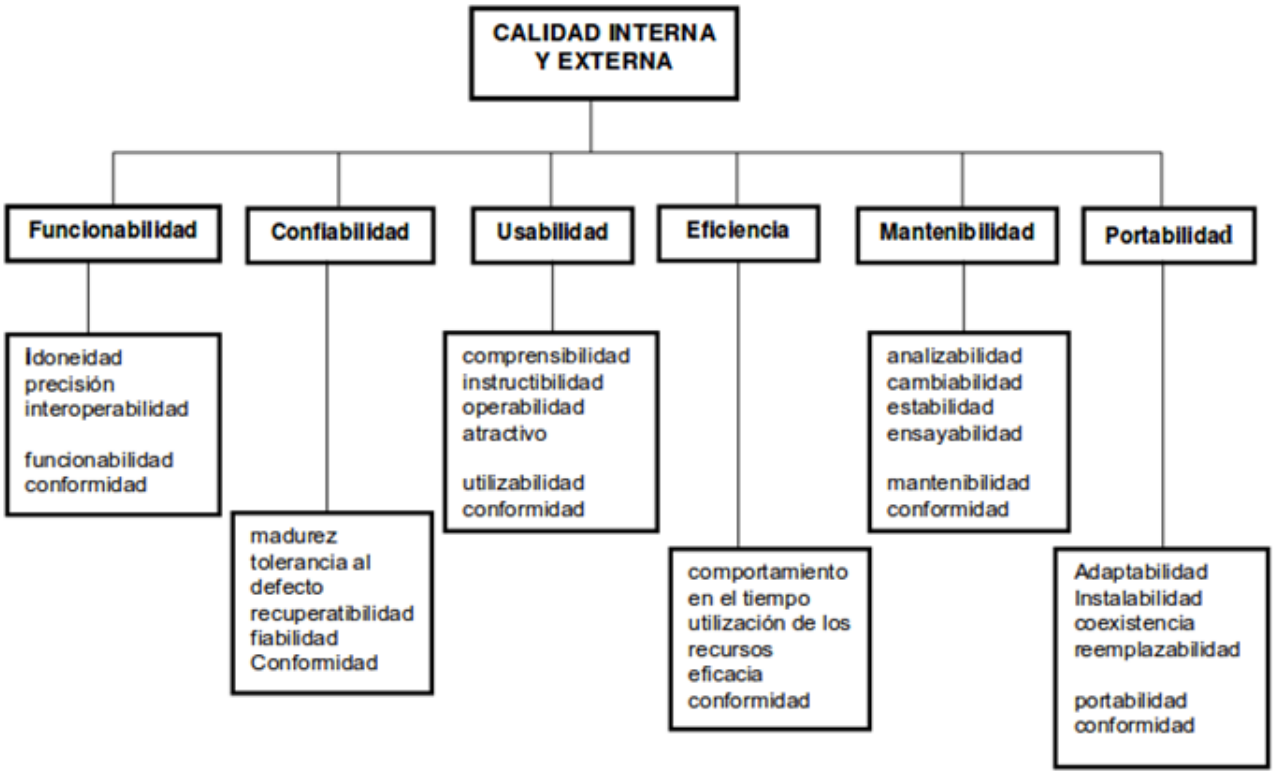

Figura 1. Clasificación de los requisitos no funcionales según la ISO 9126.

1.6 Arquitectura de desarrollo

Un patrón arquitectónico representa, en términos de componentes y las relaciones entre ellos, posibles soluciones para incorporar en el diseño, aspectos que mejoran la usabilidad del sistema final (Sommerville, 2005).

El **Modelo Vista Controlador** (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos (Modelo, Vista y Controlador). A continuación, se ofrece una breve descripción de cada una de las capas.

Modelo

Es la representación específica de la información con la cual el sistema opera. La lógica de datos

asegura la integridad de estos y permite derivar nuevos datos. Esto quiere decir que se operan los datos y las reglas de negocio asociadas al sistema, incluyendo el análisis sintáctico y el procesamiento de los datos de entrada y de los datos de salida (Fowler, 2003).

Vista

Esta presenta el Modelo, usualmente la interfaz de usuario. La vista es la capa de la aplicación que observa el usuario en un formato adecuado para interactuar, en otras palabras, es la interfaz gráfica (Fowler, 2003).

Controlador

El Controlador es la capa que controla todo lo que puede realizar la aplicación. Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Está compuesto por acciones que se representan con funciones en una clase (Fowler, 2003).

Se propone la utilización de MVC como patrón arquitectónico, para la implementación de la aplicación, ya que a través de este patrón se logra una división de las diferentes partes que conforman una aplicación, permitiendo la actualización y mantenimiento del software de una forma sencilla y en un reducido espacio de tiempo.

1.7 Artefactos de diseño

1.7.1 Patrones de diseño

Los patrones de diseño se han convertido en una técnica importante para el reúso del conocimiento de software. Cada patrón provee información sobre su diseño para describir las clases, métodos y relaciones que resuelven un problema de diseño en particular, estos han sido agrupados y organizados en catálogos cada uno para dar diferentes clasificaciones y descripciones (Erika Camacho, 2014).

Los patrones para asignar responsabilidades GRASP (General Responsibility Assignment Software Patterns, por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos en una aplicación. Se considera que más que patrones, son una serie de buenas prácticas recomendadas en el diseño de software (Rodríguez, 2014).

Experto

Este patrón posibilita que se mantenga el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Se debe tener en cuenta que esto es aplicable mientras se consideren los mismos aspectos del sistema, es decir, lógica de negocio, persistencia a la base de datos e interfaz de usuario (Larman, 2006).

Bajo acoplamiento

La función de este patrón es asignar una responsabilidad de manera que el acoplamiento permanezca bajo. Un elemento con bajo acoplamiento no depende de demasiados elementos, estos pueden ser clases, subsistemas, sistemas, etcétera. Este patrón permite que en caso de modificarse algún elemento sus cambios no afecten a otro componente, además son fáciles de entender de manera aislada y conveniente para la reutilización, lo que quiere decir que debe haber poca dependencia entre clases (Larman, 2006).

Alta cohesión

La función de este patrón es asignar una responsabilidad de manera que la cohesión permanezca alta. Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. Posibilita que se incremente la claridad, facilita la comprensión del diseño, simplifica el mantenimiento y las mejoras, soportando a menudo bajo acoplamiento y el grado bajo de funcionalidades altamente relacionadas, incrementando la reutilización (Larman, 2006).

Controlador

El patrón controlador sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón permite asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas, facilitando la centralización de actividades (validaciones, seguridad, etcétera) (Larman, 2006).

1.7.2 Herramienta CASE

Visual Paradigm 8.0: Visual Paradigm para UML es una herramienta para desarrollo de aplicaciones utilizando modelado UML ideal para ingenieros de software, analistas y arquitectos de sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos (Visual Paradigm, 2017).

Esta herramienta ofrece un conjunto completo de herramientas utilizadas por los equipos de desarrollo de software para la captura de requisitos, planificación de software, la planificación de controles, el modelado de clases y modelado de datos. Por las razones anteriores y porque es útil para la realización de diagramas de las diferentes fases del desarrollo de software se propone Visual Paradigm para el diseño de artefactos.

1.7.3 Herramienta para el diseño visual

JavaFX Scene Builder 2.0: Herramienta para el diseño visual que permite a los usuarios diseñar

rápidamente interfaces de usuario de aplicaciones JavaFX, sin codificación. Los usuarios pueden arrastrar y soltar componentes de interfaz de usuario en un área de trabajo, así como también modificar sus propiedades y aplicar hojas de estilo. Crea un código FXML del diseño que se genera automáticamente en segundo plano. El resultado es un archivo FXML que se puede combinar con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación (oracle.org).

1.7.4 Métricas de validación del diseño

Las métricas de software proporcionan una forma cuantitativa para valorar la calidad de los atributos internos de producto y, por tanto, permiten valorar la calidad antes de construir el producto. Las mismas proporcionan la comprensión necesaria para crear modelos efectivos de requerimientos y diseño, código sólido y pruebas amplias (Pressman, 2010). Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- ✓ **Responsabilidad:** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- ✓ **Complejidad de implementación:** Grado de dificultad que tiene implementado un diseño de clases.
- ✓ **Reutilización:** Grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- ✓ **Acoplamiento:** Grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de reutilización.
- ✓ **Complejidad del mantenimiento:** Grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software.
- ✓ **Cantidad de pruebas:** Número o grado de esfuerzo para realizar las pruebas de calidad del producto (componente, módulo, clase, conjunto de clases, etcétera) diseñado.

Las métricas concebidas como instrumento para evaluar la calidad del diseño y su relación con los atributos de calidad definidos son las siguientes:

Tamaño Operacional de Clase (TOC): Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad (Mark Lorenz, 1994).

Atributos	**Modo en que lo afecta**
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad

	asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Tabla 1. Atributos de calidad y modo en que afectan al instrumento TOC.

Los criterios y categorías definidos para la evaluación de los atributos de calidad anteriores se presentan en la siguiente tabla:

Atributo	**Categoría**	**Criterio**
Responsabilidad	Baja	< = Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Complejidad de implementación	Baja	< = Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<= Promedio

Tabla 2. Criterios de evaluación para la métrica TOC.

Relaciones entre Clases (RC): Está dado por el número de relaciones de uso de una clase con otra (Mark Lorenz, 1994).

Atributos	**Modo en que lo afecta**
Acoplamiento	Un aumento del RC implica un aumento del acoplamiento de la clase.

Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 3. Atributos de calidad y modo en que afectan al instrumento RC.

Atributo	**Categoría**	**Criterio**
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de mantenimiento	Baja	< = Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<= Promedio
Cantidad de pruebas	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Tabla 4. Criterios de evaluación para la métrica TOC.

1.8 Sistema Gestor de Base de Datos (SGBD)

A continuación, se describen las características y ventajas de las diferentes bases de datos embebidas estudiadas.

SQLite

SQLite es una herramienta de software libre, sin dependencias externas y multiplataforma que permite almacenar información en dispositivos embebidos de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware. No es necesario configurar ni administrar. Almacena todo el contenido de la base de datos en un solo fichero. Se puede utilizar desde C y C++, pero también desde otros lenguajes de programación (PHP, Python, Java, .NET). En el caso de Java permite gestionar la base de datos a través de JDBC (sqlite.org) (Sánchez, 2014).

Derby o JavaDB

JavaDB es una distribución de Oracle de la base de datos de código libre Derby. Soporta el estándar ANSI/ISO SQL a través de JDBC y Java EE. Estas librerías están incluidas en el JDK. Almacena la base de datos en múltiples archivos, lo que puede resultar útil para escalar el almacenamiento. Solo se puede utilizar en Java, no desde otros lenguajes (db.apache.org) (Sánchez, 2014).

ObjectDB

Es una base de datos orientada a objetos que permite acceso JPA, un estándar de Java que persigue no perder las ventajas de la orientación a objetos; esto se suele perder cuando se acerca a la capa de persistencia. Esta base datos es completa y fácil de usar. ObjectDB está protegida contra transacciones y recuperable por fallos (objectdb.org) (Sánchez, 2014).

La siguiente imagen expresa el comportamiento de las bases datos expuestas anteriormente según el rendimiento al realizar las consultas SQL.

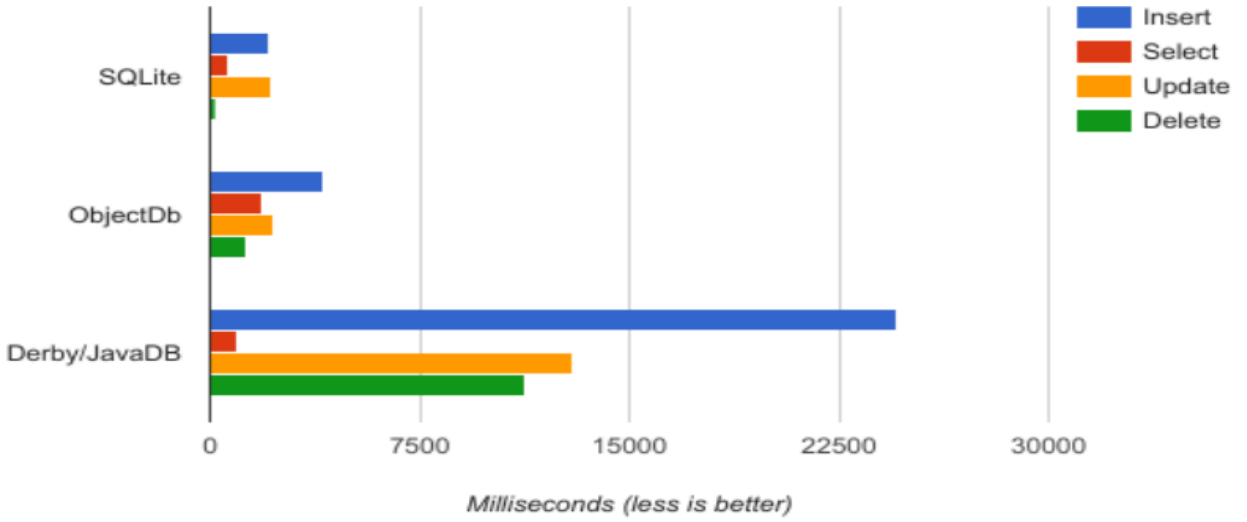

Figura 2. Comparación de rendimiento de las bases datos embebidas Java. (Sánchez, 2014)

Luego de analizar lo expuesto anteriormente se determina utilizar SQLite como SGBD, ya que permite el almacenamiento local/cliente con mejor rendimiento en aplicaciones software tales como navegadores, sistemas operativos, sistemas embebidos y de escritorios.

1.9 Lenguaje de programación

JavaFX 8.0: Es un lenguaje de programación basado en Java que permite crear aplicaciones de escritorio. Posee las siguientes características (Lowe, 2014):

- Multiplataforma, desde el escritorio hasta los dispositivos móviles.
- Utiliza el mismo lenguaje para la web, para el escritorio y para la telefonía móvil.
- Permite a los desarrolladores integrar gráficos vectoriales, animación, sonido y activos web de vídeo en una aplicación interactiva, completa y atractiva.
- Amplía la tecnología Java permitiendo el uso de cualquier biblioteca de Java en una aplicación JavaFX.
- Permite mantener un eficaz flujo de trabajo entre diseñador y desarrollador en el que los diseñadores pueden trabajar en las herramientas que deseen mientras colaboran con los desarrolladores.

Se utiliza este lenguaje de programación, ya que el lenguaje JavaFX permite que la aplicación sea ejecutada en una computadora local, aunque esté desconectada de Internet, posibilitando que sea más veloz cuando sea compilada dicha aplicación.

1.10 Entorno Integrado de Desarrollo (IDE)

NetBeans 8.2: Es un entorno de desarrollo visual de código abierto para aplicaciones programadas mediante Java, uno de los lenguajes de programación más poderosos del momento. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones web, o para dispositivos móviles. Brinda soporte a las siguientes tecnologías, entre otras: Java, PHP, C/C++, HTML5**.** Además, puede instalarse en varios sistemas operativos: Windows, Linux, entre otros (netbeans.org).

1.11 Pruebas de software

El proceso de pruebas es un elemento fundamental de la metodología AUP-UCI, este reduce el tiempo de desarrollo y así la cantidad de tiempo necesario para integrar y estabilizar versiones. Mejora la productividad encontrando y arreglando errores rápidamente, además de incrementar la calidad global del software garantizando que todo el código nuevo ha sido probado, y a todo el código existente se le aplican pruebas de regresión antes de ser integrados a la base central de código. Dicha metodología divide las pruebas en tres grupos entre los que se encuentran:

Pruebas Unitarias: Son las pruebas implementadas por los desarrolladores, encargadas de verificar el código, es decir, el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (Universidad de las Ciencias Informáticas, 2015).

Pruebas de Aceptación (o pruebas funcionales): Es la prueba final antes del despliegue del sistema. Son las pruebas destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente (Universidad de las Ciencias Informáticas, 2015).

1.11.1 Pruebas Unitarias

Las pruebas de unidad o unitarias son una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

Es la mejor forma de detectar errores en tempranas etapas de desarrollo y ayudan a definir requerimientos y responsabilidades de cada método en cada clase. Aunque no garantizan detectar todos los errores, por tanto, deben ejecutarse en conjunto con otras pruebas (Pressman, 2010).

Método de prueba Caja Blanca

La prueba de caja blanca del software se basa en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles (Pressman, 2010).

Dentro de la prueba de caja blanca se incluyen las Técnicas de Pruebas que serán descritas a continuación:

- **Prueba del Camino Básico:** Permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la definición de un conjunto de caminos básicos.
- **Prueba de Condición:** Ejercita las condiciones lógicas contenidas en el módulo de un programa. Garantiza la ejecución por lo menos una vez de todos los caminos independientes de cada módulo, programa o método.
- **Prueba de Flujo de Datos:** Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa. Garantiza que se ejerciten las estructuras internas de datos para asegurar su validez.
- **Prueba de Bucles:** Se centra exclusivamente en la validez de las construcciones de bucles. Garantiza la ejecución de todos los bucles en sus límites operacionales.

1.11.2 Pruebas de Aceptación

Las pruebas de aceptación son pruebas de caja negra definidas por el cliente, estas tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas.

Método de prueba de Caja Negra

Las pruebas de caja negra, también llamadas pruebas de comportamiento La prueba de caja negra se refiere a las pruebas que se llevan a cabo en la interfaz del software. Una prueba de caja negra examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software. (Pressman, 2010)

En esencia intentan encontrar errores en las categorías siguientes:

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores en las estructuras de datos o en el acceso a bases de datos externas.
- Errores de comportamiento en inicialización y terminación.

Algunos de los métodos empleados en las pruebas de Caja Negra son los siguientes:

- Métodos de prueba basados en grafos.
- Partición de equivalencia.
- Análisis de valores límite.
- Prueba de la tabla ortogonal.

1.12 Conclusiones parciales del capítulo

Luego de desarrollar el capítulo se ha podido arribar a las siguientes conclusiones parciales:

- El análisis de los conceptos de fuera de línea e ingresos y de los aspectos teóricos relacionados con la gestión de almacenamiento posibilitó un mayor entendimiento del tema investigado.
- El estudio de las técnicas de almacenamiento fuera de línea brindó algunos beneficios para el desarrollo del componente Ingresos, proporcionando nuevas ideas sobre el funcionamiento interno de la base datos de una aplicación de escritorio.
- Con el estudio de la metodología de desarrollo de software AUP-UCI se logró definir el lenguaje y las herramientas más adecuadas, así como los artefactos necesarios para dar solución a la investigación.

Capítulo 2: Análisis y diseño del sistema.

2.1 Introducción

En este capítulo se determina la propuesta de solución, proporcionando un mejor entendimiento del componente. Se describen brevemente los requisitos funcionales y no funcionales previamente identificados. Se muestra un modelo de datos con su respectivo diccionario de datos para comprender el entorno en el que trabaja el componente. Además, se expone la arquitectura, los patrones para el diseño del sistema y el resultado de las métricas para validar el diseño del componente.

2.2 Descripción del sistema a desarrollar

El sistema permite adicionar, listar, modificar y eliminar principalmente los ingresos, así como las comisiones adicionales y los ajustes de comisión de agentes de la ESEN. Además, posibilita gestionar los pagos prima correspondientes a los ingresos.

El sistema cuenta con una base datos embebida donde se almacena la información necesaria para registrar los ingresos y las comisiones de los agentes. El cliente interactúa con el sistema a través de la vista, el sistema resuelve las peticiones mediante la relación del modelo y el controlador, este devuelve la información resultante y el sistema muestra la interfaz al cliente. El acceso al sistema se realiza mediante una aplicación de escritorio JavaFX.

2.3 Requisitos funcionales analizados

A continuación, se listan los requisitos funcionales del componente previamente identificados por equipo de desarrollo del proyecto.

Nº	Requisitos	Prioridad para el cliente	Complejidad
1	Registrar ingresos	Alta	Alta
2	Buscar ingreso	Alta	Alta
3	Liquidar ingreso	Alta	Alta
4	Listar ingresos	Baja	Baja
5	Registrar pagos prima	Baja	Baja
6	Listar pagos prima	Baja	Baja
7	Registrar otros pagos de agentes	Media	Media

8	Registrar ajuste de comisión de agentes	Media	Media

Tabla 5. Requisitos funcionales.

2.3.1 Descripción de requisito por procesos

A continuación, se realiza la especificación del requisito Registrar ingresos. Para consultar las restantes descripciones de los requisitos por procesos, revisar anexo 1.

RF1 Registrar ingresos

Precondiciones	El usuario se ha autenticado en el sistema y cuenta con los permisos necesarios para ejecutar esta acción. El usuario selecciona la opción Gestionar Ingresos/Registrar ingreso. Existe al menos una póliza registrada en el sistema.
Flujos de eventos	
Flujo básico Registrar ingresos por póliza	
1.	Se muestran en la ventana Gestionar Ingresos los siguientes campos para que el usuario introduzca los datos (Ver validación 1 y 2): **Nro. Documento de banco** (número del documento de banco donde consta el depósito del ingreso en el banco). **Importe documento de banco** (importe del documento de banco). **Importe comisión de banco** (importe de la comisión de la operación de banco que aparece en el documento que emite). Se muestra además en interfaz el **Importe del ingreso** (constituye la suma de los importes que se registran asociados a las pólizas por cualquier motivo que se haya seleccionado, se va actualizando).
2.	Se muestra el motivo de ingreso **Pago de prima** Ver flujo alternativo 2.a Registrar pagos de primas.
3.	Se presiona el botón **Adicionar**.
4.	Se validan los datos introducidos (Ver validación 4).
5.	Si los datos introducidos son correctos se registran los datos del ingreso guardando el ingreso en estado En tránsito.

6.	Se muestra un mensaje de información indicando que la acción se ha realizado satisfactoriamente.
7.	Concluye el requisito.
Poscondiciones	
1.	Se guardó el ingreso en estado En tránsito.
2.	Se limpiaron los campos de la interfaz para el registro de otro ingreso.
3.	Se muestra un listado actualizado de los ingresos pólizas que están en estado En tránsito en el sistema. (Ver requisito Listar ingresos de esta agrupación de requisitos).
Flujos alternativos	
Flujo alternativo 2.a Registrar pagos de prima	
1.	Se listan, si hay, los registros de pagos primas realizados por el usuario en sesión que no se han liquidado (Ver requisito Listar pagos prima de esta agrupación de requisitos), es decir, que no han sido incluidos en el registro de ningún ingreso y se da al usuario también la posibilidad de registrar pagos prima.
2.	Si el usuario presiona Pagos prima se ejecuta el requisito Registrar pago prima de esta misma agrupación de requisitos, sino continuar al paso 3 de este flujo alternativo
3.	El usuario selecciona uno o varios registros pagos prima de los que se listan.
4.	Volver al paso 3 del flujo básico.
Poscondiciones	
1.	Se seleccionaron uno o varios registros de pagos de prima no liquidados.
Validaciones	
1	Se validan los datos según lo establecido en el Modelo de datos (Figura 2).
2	El Importe comisión de banco no puede ser mayor que el Importe documento de banco.
3	El sistema valida que se encuentra registrada en el sistema la póliza y que no está en estado

	Cancelada.
4	El sistema comprueba que el importe del documento de banco coincida con el importe total del ingreso (la suma de los importes de los pagos primas seleccionados).

Prototipo elemental de interfaz gráfica de usuario

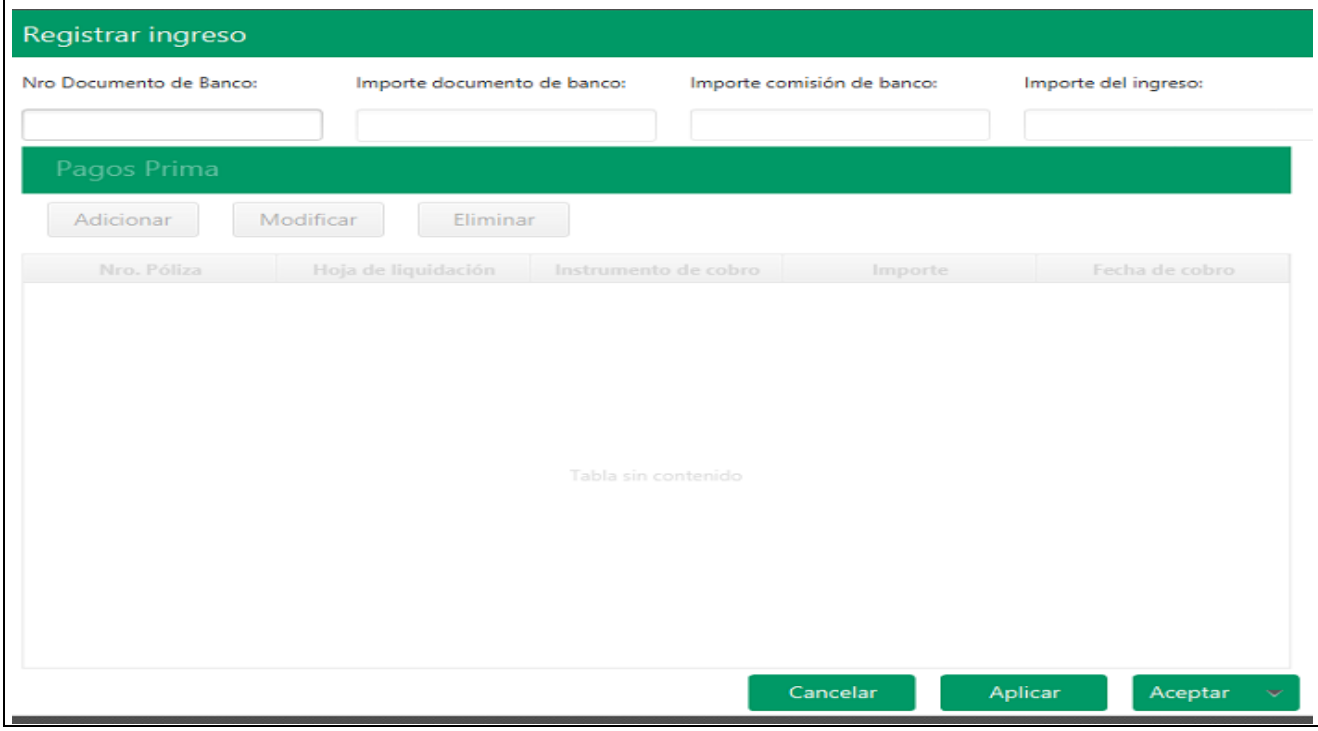

Tabla 6. Descripción del requisito Registrar ingreso.

2.4 Requerimientos no funcionales

A continuación, se describen los requisitos no funcionales del componente, haciendo uso de los atributos de calidad mencionados en el capítulo 1.

Nº	Requisitos	Atributo de Calidad
1	Garantizar la disponibilidad de la aplicación cuando no exista conectividad con los servidores centrales.	Portabilidad
2	Responder al 100% de los criterios seleccionados en las búsquedas.	Funcionabilidad
3	Mostrar en los mensajes de error los datos identificativos del lugar, operación o dato con error.	Funcionabilidad
4	Permitir solo la entrada de datos correctos en los formularios	Confiabilidad

	requeridos.	
5	Prevenir la eliminación de información de la base de datos que se haya asociado a alguna operación.	Confiabilidad
6	Mostrar mensajes de error ante el fallo de una funcionalidad del sistema indicando la causa.	Confiabilidad
7	Asegurar acceso al sistema luego de autenticarse mediante usuario y contraseña válidos.	Seguridad
9	Permitir acceder a todas las funcionalidades para entrar los datos y procesar la información a través de un menú.	Usabilidad
10	Permitir guardar actualizaciones en el sistema luego de restablecerse la conexión con los servidores centrales.	Funcionalidad

Tabla 7. Requerimientos no funcionales.

2.5 Diseño del sistema

El diseño de software es un proceso iterativo por medio del cual se traducen los requerimientos en un plano para construir el software. El diseño se representa en un nivel alto de abstracción, en el que se rastrea directamente el objetivo específico del sistema y los requerimientos más detallados de datos, funcionamiento y comportamiento (Pressman, 2010).

2.5.1 Diagrama de clases del diseño

El diagrama de clases del diseño se realiza para describir las especificaciones de las clases y las interfaces de una aplicación. A continuación, se muestra el diagrama de clases del diseño del requisito Registrar ingreso, reflejado en el funcionamiento de la arquitectura MVC.

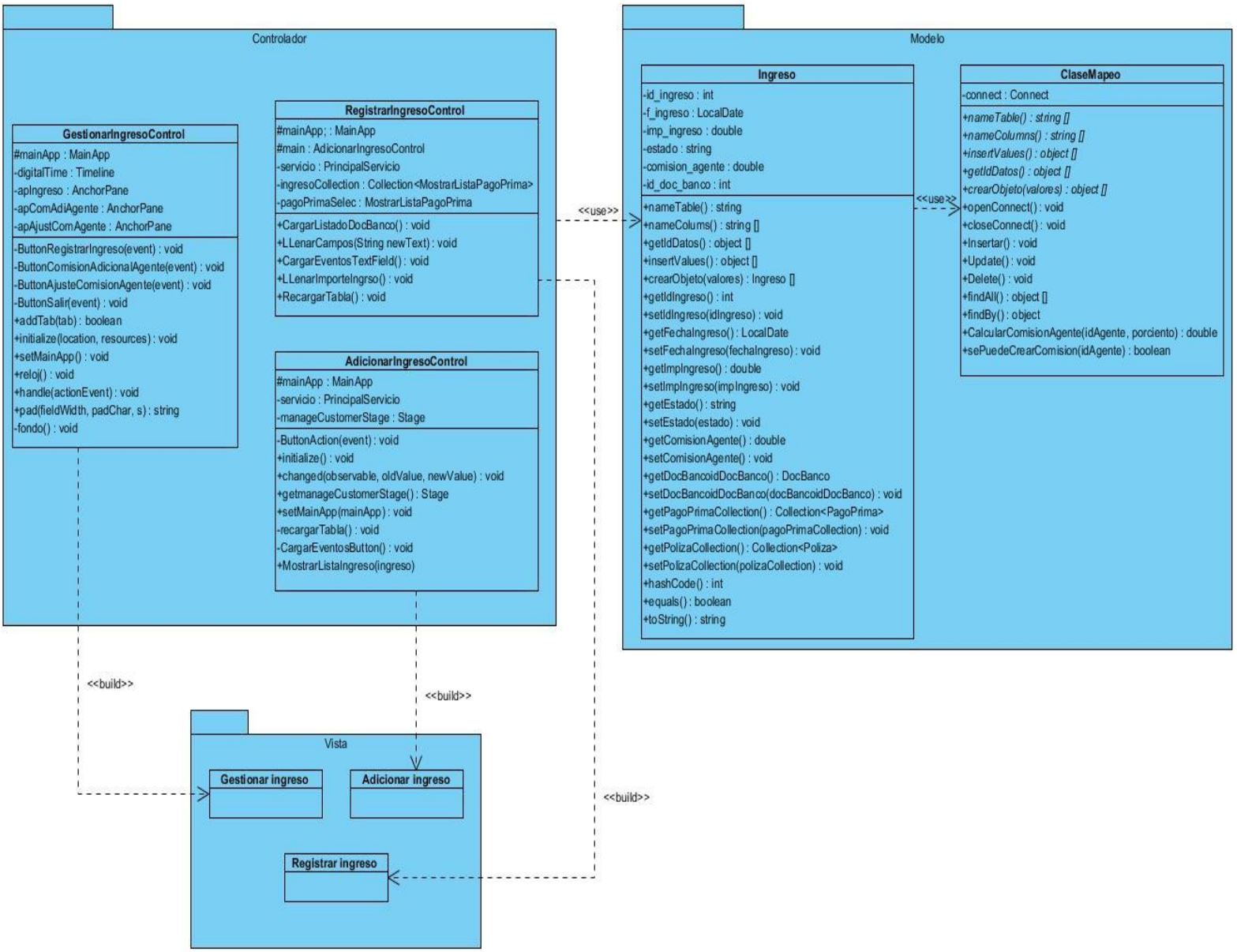

Figura 3. Diagrama de clases del diseño del requisito Registrar ingreso.

2.5.2 Modelo de datos (MD)

La mayoría de los sistemas de software grandes utilizan base de datos de información de gran tamaño. En algunos casos, esta base de datos es independiente del sistema de software, en otros, se crea para el sistema que se está desarrollando. Una parte importante del modelado de sistemas es la definición de la forma lógica de los datos procesados por el sistema.

La técnica de modelado de datos más ampliamente utilizada es el diagrama entidad-relación, que representa todos los datos que se introducen, almacenan, transforman y generan dentro de una aplicación. Además, muestra las entidades de datos, sus atributos asociados y las relaciones entre estas entidades (Sommerville, 2005) (Pressman, 2010).

A continuación, se observa el modelo de datos correspondiente a la solución propuesta:

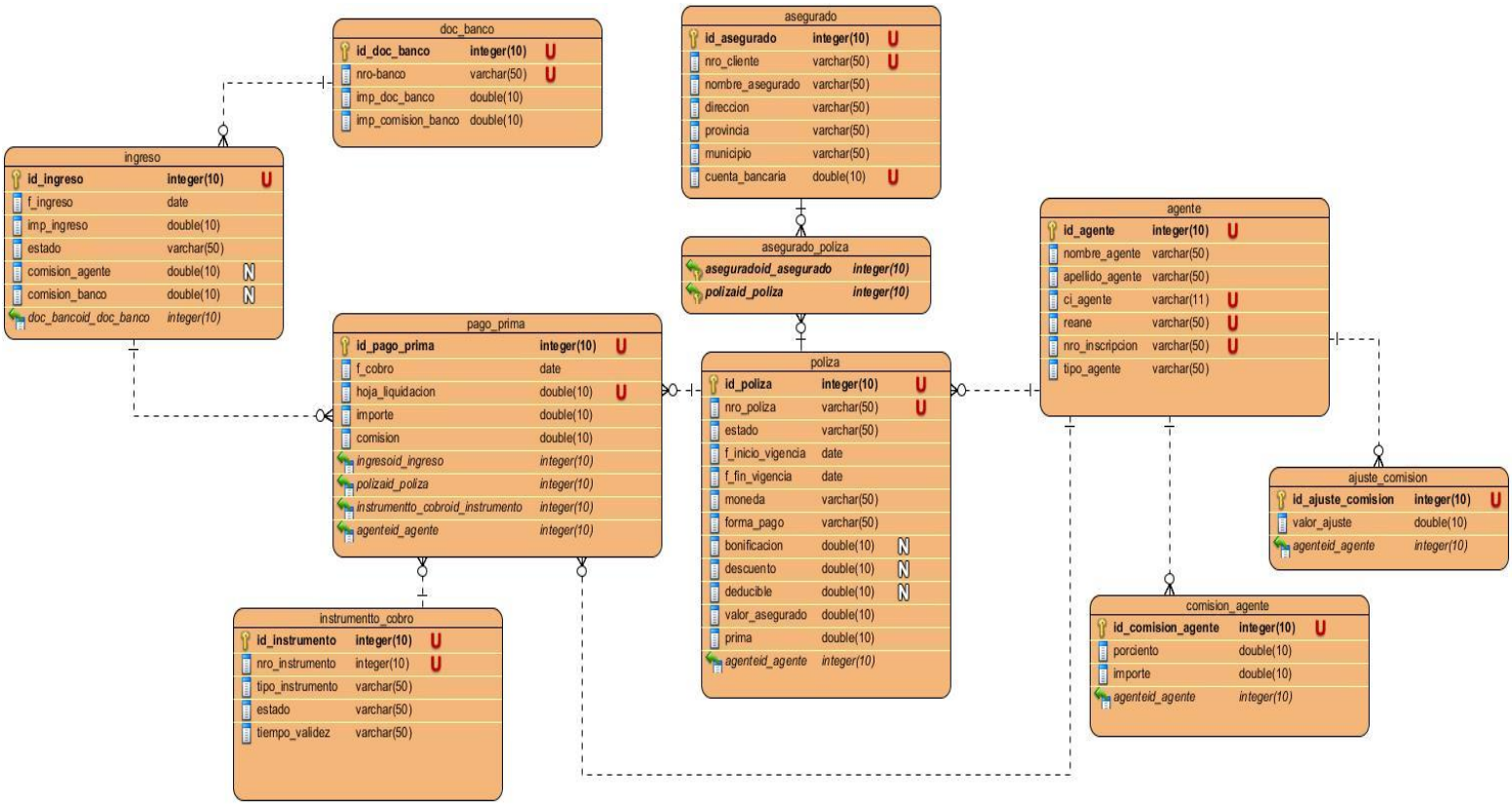

Figura 4. Modelo de datos del componente.

En el MD se puede apreciar la tabla *ingreso* que contiene sus atributos para registrar un ingreso, manteniendo una relación de UnoAMuchos con la tabla *pago_prima*, la cual permite registrar a partir de un determinado número de documento de banco la cantidad de pagos primas asociados a determinado ingreso. Se muestra la tabla *agente* relacionada de UnoAMuchos con las tablas *poliza*, *pago_prima*, *comisión_agente* y *ajuste_comision_agente*, donde las dos últimas guardan los datos relacionados con las comisiones según el tipo agente. Además, se puede observar la relación de MuchosAMuchos de la tabla *poliza* con la tabla *asegurado*, donde la primera es la encargada de guardar una lista de las pólizas que están comercializadas y la segunda contiene los datos de los asegurados asociados a determinada póliza.

Diccionario de datos

La siguiente tabla describe cada atributo de la entidad Ingreso, para observar las demás descripciones revisar anexo 2.

Ingreso

Descripción	Importe que se ingresa a la entidad por varios conceptos.
Atributos	

Nombre	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clases válidas	**Clases no válidas**
Fecha	Fecha del ingreso	Fecha	No	Si	Números y el carácter "-"	
Estado	Estado del ingreso	Cadena de caracteres	No	No	En tránsito Liquidado	
Importe	Es el valor que se le planifica en un plan a un indicador	Decimal	No	No	Números	
Comisión de agente	Parte del ingreso que se asigna a un agente	Decimal	Si	No	Números	
Comisión de banco	Parte del ingreso que se asigna al banco	Decimal	Si	No	Números	

Tabla 8. Diccionario de datos de la entidad Ingreso.

2.5.3 Descripción de la arquitectura

La arquitectura no es el software operativo, es una representación que permite analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reducir los riesgos asociados con la construcción del software. El estándar IEEE define una descripción arquitectónica como "un conjunto de productos para documentar una arquitectura" (Pressman, 2010).

Para el sistema a desarrollar el patrón de arquitectura de software es Modelo Vista Controlador (MVC) que se estructura en tres capas: Modelo, Vista y Controlador. En el **modelo** se encuentran las clases con funciones para consultar y transformar la información de la base de datos, la cual se transfiere hacia el **controlador** donde se aplica toda la lógica de negocio, se procesa y se envían los datos a la vista. En el sistema la **vista** está compuesta por clases ". *fxml"* que se encargan de mostrar la información al usuario. En el controlador se encuentran aquellas clases etiquetadas *@Control* que

procesan las peticiones de los usuarios, realizan las operaciones con las clases del modelo y conforman la vista que será mostrada. En el modelo se encuentra la clase *"ClaseMapeo"* responsable de enviar y recibir información de la base datos, procesarla y entregarla al controlador.

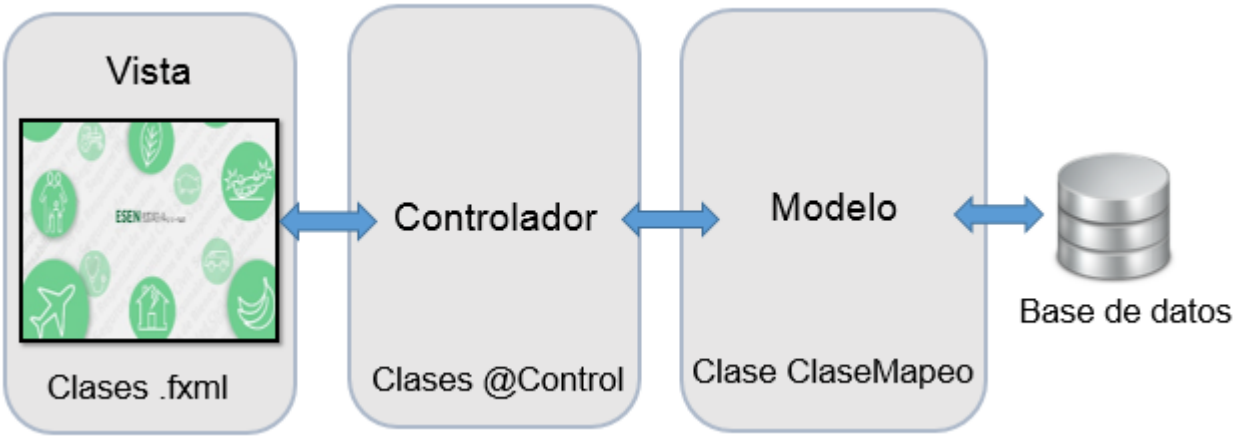

Figura 5. Funcionamiento del patrón MVC en el sistema.

2.5.4 Uso de los patrones de diseño

A continuación, se describen los patrones GRASP presentes en el sistema.

Experto

Este patrón se evidencia por ejemplo en la clase *Ingreso*, la misma posee toda la información para crear los ingresos. La siguiente imagen muestra el código del método *crearObjeto ()* de dicha clase.

```
@Override
public Ingreso[] crearObjeto(String[] valores) {
    int x = valores.length;
    Ingreso[] nuevosObjetos = new Ingreso[x];
    for (int i = 0; i < valores.length; i++) {
        String val = valores[i];
        String[] propiedades = val.split(",");
        Ingreso newIngreso = new Ingreso();
        newIngreso.setIdIngreso(Integer.parseInt(propiedades[0]));
        newIngreso.setFechaIngreso(LocalDate.parse(propiedades[1]));
        newIngreso.setImpIngreso(Double.parseDouble(propiedades[2]));
        newIngreso.setEstado(propiedades[3]);
        if (propiedades[4].equals("null")) {
            newIngreso.setComisionAgente(null);
        } else {
            newIngreso.setComisionAgente(Double.parseDouble(propiedades[4]));
        }
        if (propiedades[5].equals("null")) {
            newIngreso.setComisionBanco(null);
        } else {
            newIngreso.setComisionBanco(Double.parseDouble(propiedades[5]));
        }
        newIngreso.setDocBancoidDocBanco(Integer.parseInt(propiedades[6]));
        nuevosObjetos[i] = newIngreso;
    }
    return nuevosObjetos;
}
```

Figura 6. Ejemplo del patrón Experto.

Bajo acoplamiento

La utilización de este patrón se evidencia en todas las clases del modelo que utilizan un método que devuelve los valores a insertar, de este modo la modificación que se realice en una clase no afectaría a las otras. La siguiente imagen muestra el código del método *insertValues ()* de la clase *Ingreso*, donde se evidencia lo mencionado anteriormente.

```
@Override
Object[] insertValues() {
    Object[] valores = {idIngreso, fechaIngreso, impIngreso, estado, comisionAgente, comisionBanco, docBancoidDocBanco.getIdDocBanco()};
    return valores;
}
```

Figura 7. Ejemplo del patrón Bajo acoplamiento.

Alta cohesión

La clase *ClaseMapeo* presenta la funcionalidad de *crearObjeto ()* (Figura 5) y colabora con las clases del Modelo (para comprobar que la información enviada desde la base datos es correcta y devolver los objetos del tipo correspondiente) para obtener la información correctamente.

Controlador

Algunas de las clases que implementan este patrón son: *GestionarIngresoControl*, *AdicionarIngresoControl, AdicionarAjusteDeComisiónControl*, *AdicionarPagosPrimaControl*. En la siguiente imagen se evidencia un ejemplo del método *CargarEventosButton ()* la función *setOnAction* del botón Liquidar que aparece en el controlador *AdicionarIngresoControl,* el cual permite liquidar un ingreso seleccionado.

```
private void CargarEventosButton() {
    BUTTON_Adicionar.setOnAction((ActionEvent t) -> {
        try {
            FXMLLoader loader = servicio.lanzarStage("Registrar ingreso.fxml");
            RegistrarIngresoControl registrarIngreso = loader.getController();
            registrarIngreso.setMainAppPadre(this);
        } catch (IOException ex) {
            Logger.getLogger(AdicionarIngresoControl.class.getName()).log(Level.SEVERE, null, ex);
        }
    });
    BUTTON_Liquidar.setOnAction((ActionEvent t) -> {
        Ingreso ingresoAux = new Ingreso();
        Ingreso[] ArrayIngreso = (Ingreso[]) ingresoAux.findBy("id_ingreso", ingresoSelec.getIdIngreso());
        Ingreso ingreso = ArrayIngreso[0];
        ingreso.setEstado("Liquidado");
        ingreso.Update();
        servicio.CrearAlertaInformacion("Se ha realizado correctamente la liquidación del ingreso.");
        recargarTabla();

    });
```

Figura 8. Ejemplo del patrón Controlador.

2.5.5 Resultado de la aplicación del instrumento de evaluación de la métrica Tamaño Operacional de Clase (TOC)

El instrumento de evaluación TOC fue aplicado a una muestra de 14 clases escogidas aleatoriamente del componente Ingresos, la Figura 9 muestra el porciento de la cantidad de procedimientos presentes en las clases agrupados en intervalos definidos, resultando 2 clases entre 1-5 procedimientos, 2 clases entre 6-10 procedimientos, 6 clases entre 11-15 procedimientos y 3 clases entre 16-20 procedimientos.

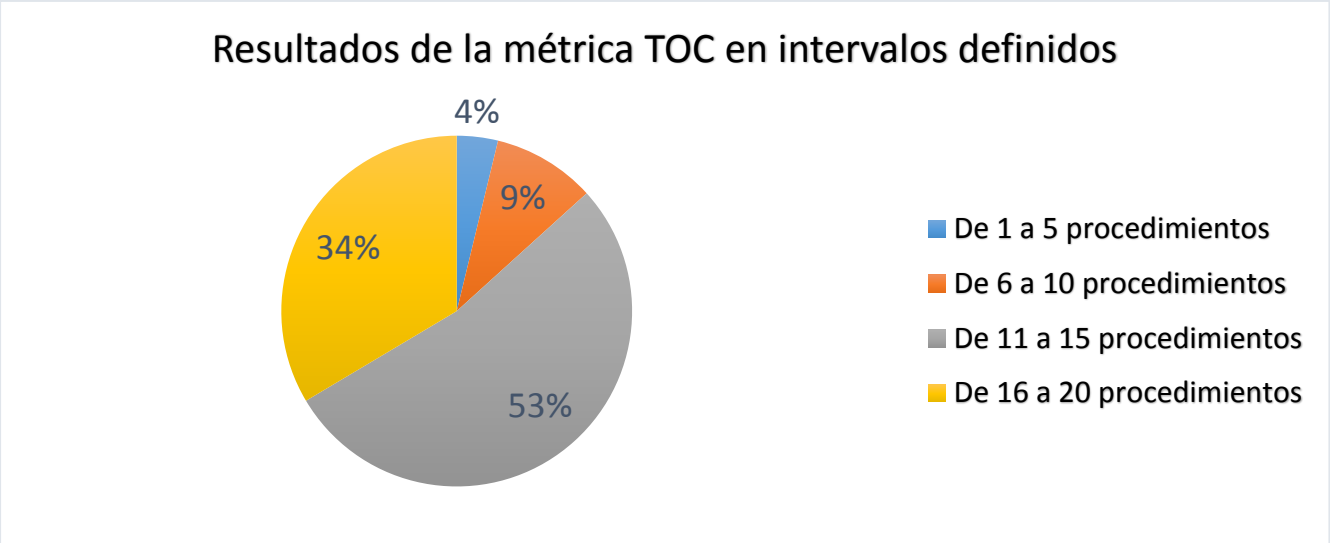

Figura 9. Representación en porciento agrupado en intervalos definidos de los resultados obtenidos tras aplicar el instrumento TOC.

Figura 10. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

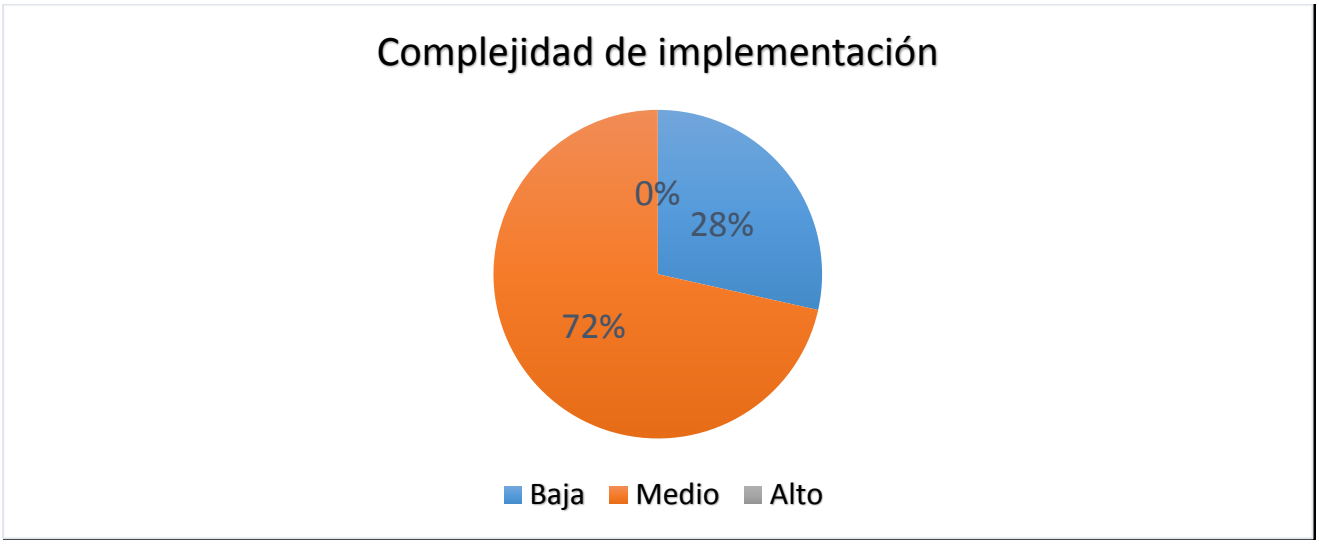

Figura 11. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad.

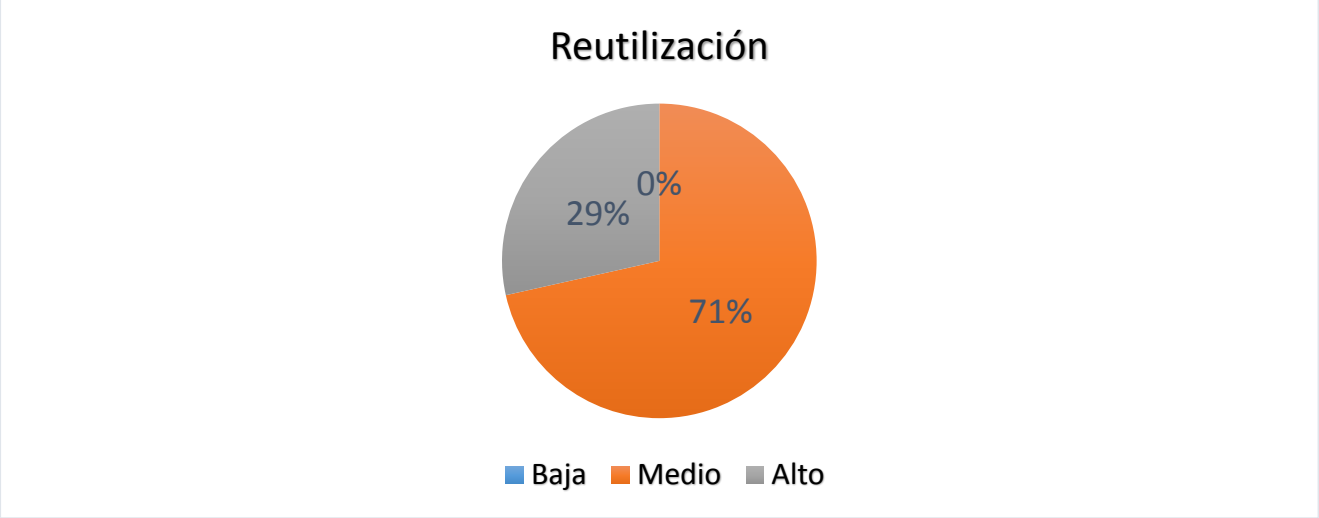

Figura 12. Representación de la incidencia de los resultados de la evaluación del instrumento TOC en el atributo Reutilización.

Al hacer un análisis de los resultados obtenidos se concluye que la mayoría de las clases incluidas poseen de 11 a 15 procedimientos en su composición representando el 50% de la muestra. Se puede observar que la mayoría de las clases que conforman el sistema para los atributos responsabilidad y complejidad están dentro de la categoría Media y Baja para un 72% y 28% respectivamente mientras que el atributo Reutilización cuenta con similares porcientos en las categorías Media y Alta mostrando así que el componente cuenta con un buen grado de reutilización, baja complejidad y responsabilidad en el diseño propuesto. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

2.5.6 Resultado de la aplicación del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

El instrumento de evaluación RC fue aplicado a una muestra de 8 clases del componente Ingresos seleccionadas aleatoriamente, la Figura 13 ilustra la cantidad de dependencias y el porciento que representan de la muestra seleccionada. La figura 14 muestra los resultados que arrojó la aplicación de la métrica en los atributos de calidad acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas.

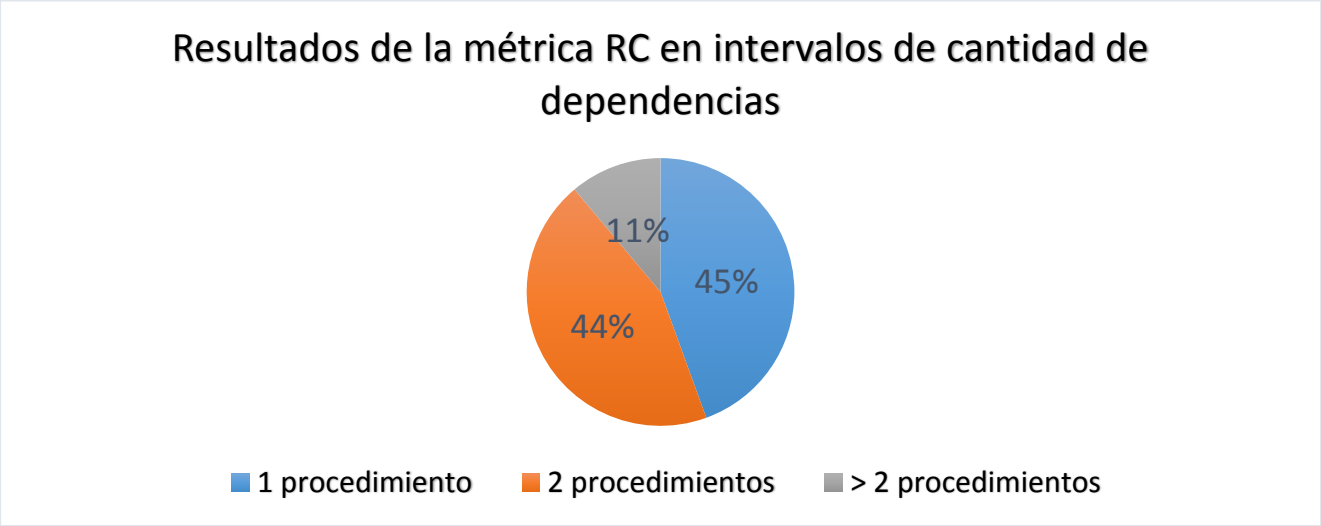

Figura 13. Representación en porciento de la aplicación del instrumento RC en intervalos definidos.

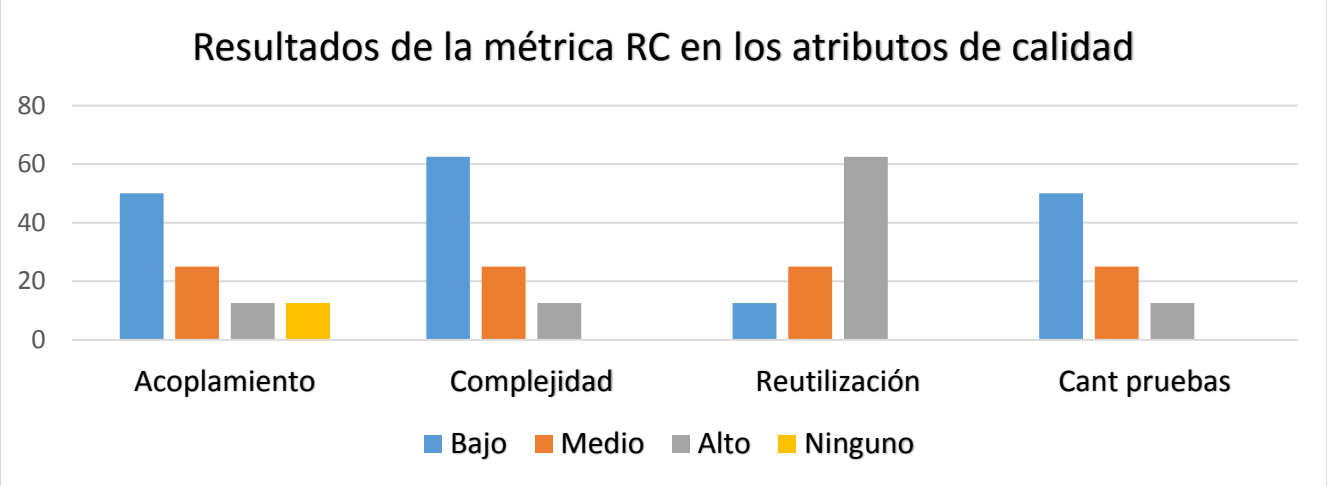

Figura 14. Representación de la incidencia de los resultados de la evaluación del instrumento RC en los atributos de calidad.

Al realizar un análisis de los resultados obtenidos a través de la métrica de software RC aplicada se obtiene que el 50% de las clases incluidas poseen bajo acoplamiento lo cual ayuda significativamente en el sentido de que al ser afectada una clase este cambio no repercutirá en las demás. El porciento

de reutilización de clases (63% aproximadamente) es alto lo cual fomenta y potencia el uso de las mismas en otras clases, en caso de ser necesario, además la complejidad de mantenimiento es en un 63% aproximadamente baja, facilitando que no resulte difícil el mantenimiento de las mismas (por ejemplo la optimización de métodos) y por último el atributo relacionado con la cantidad de pruebas es 50% bajo lo cual señala que a las clases se les puede realizar un bajo número de pruebas de poca complejidad y de bajos costes.

2.6 Conclusiones parciales del capítulo

Con el desarrollo de este capítulo se obtuvo como resultado las siguientes conclusiones:

- La descripción de la propuesta de solución permitió entender el objetivo y funcionamiento del componente Ingresos.
- Con el análisis y descripción de los 8 requisitos funcionales y los 10 no funcionales previamente determinados, se logró realizar prototipos de interfaz de usuario a través de la herramienta SceneBuilder y adaptar estos a la solución de escritorio para facilitar la implementación del componente.
- La realización de las métricas TOC y RC arrojó resultados satisfactorios. Dichas métricas permitieron determinar que aproximadamente más del 50% de las clases del sistema poseen baja complejidad, reutilización y acoplamiento, lo cual significa que se logró un buen diseño del sistema.

Capítulo 3: Implementación y pruebas.

3.1 Introducción

En el presente capítulo se generan los artefactos correspondientes a las fases de implementación y pruebas. En la etapa de implementación se describen los estándares de codificación empleados durante la implementación del componente, con el propósito de facilitar su comprensión. En la etapa de prueba se comprueba el correcto funcionamiento del software, realizan las pruebas pertinentes para garantizar la calidad y eficiencia del sistema y se exponen los resultaos obtenidos.

3.2 Fase de implementación

La etapa de implementación del desarrollo de software es el proceso de convertir una especificación del sistema en un sistema ejecutable (Sommerville, 2005).

3.2.1 Estándar de codificación

Se definen estándares de codificación porque un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible (Calleja, 2017).

Comentarios

Los comentarios deben añadir claridad al código. Deben ser concisos e idealmente hay que escribir la documentación antes que el código (Calleja, 2017). Son comentarios explicativos de una parte pequeña del código. Sintaxis: //... o /*... */.

Declaraciones de clases e interfaces

En las declaraciones de clases e interfaces no hay espacio entre el nombre del método, el paréntesis y la lista de parámetros. Se abre la llave “{“al final de la misma línea que la declaración y la llave de “}” debe aparecer en línea aparte con la misma indentación que el método o clase que cierra (Calleja, 2017).

Asignación de nombres

Cada tipo de elemento debe nombrarse con una serie de reglas determinadas:

- Paquetes: Todo en minúsculas. Cada palabra es más específica que la anterior.
- Clases e interfaces: Nombres. La inicial en mayúscula.
- Métodos: Deben ser verbos. La primera letra de la primera palabra en minúsculas, el resto de las palabras empiezan por mayúsculas.

- Variables: Deben comenzar por minúscula. No se utilizará en ningún caso el caracter "_".
- Constantes: Todo en mayúscula. Si son varias palabras, unidas por el caracter "_".

En las siguientes figuras se muestra el estilo de codificación del método *recargarTabla ()* perteneciente al controlador *AdicionarIngresoControl*, el cometario referente al objetivo de dicho método y la estructura de los paquetes empleados en la implementación del componente.

```
/* Estructura los datos y recarga la tabla de los ingresos */
    private void recargarTabla() {
        Ingreso ingresoAux = new Ingreso();
        valIngreso = (Ingreso[]) ingresoAux.findAll();
        listaMostrarIngreso = new ArrayList<>();
        MostrarListaIngreso x;
        for (Ingreso ingreso : valIngreso) {
            x = new MostrarListaIngreso(ingreso);
            listaMostrarIngreso.add(x);

        }
        ingresosArrayList = FXCollections.observableArrayList();
        ingresosArrayList.clear();
        ingresosArrayList.addAll(listaMostrarIngreso);
        tablaIngreso.setItems(ingresosArrayList);
        if (tablaIngreso.getSelectionModel().isEmpty()) {
            BUTTON_Liquidar.setDisable(true);
            BUTTON_ImpDumento.setDisable(true);
            BUTTON_ImpResumen.setDisable(true);
        }
    }
```

Figura 15. Código del método *recargarTabla* () referente a los ingresos.

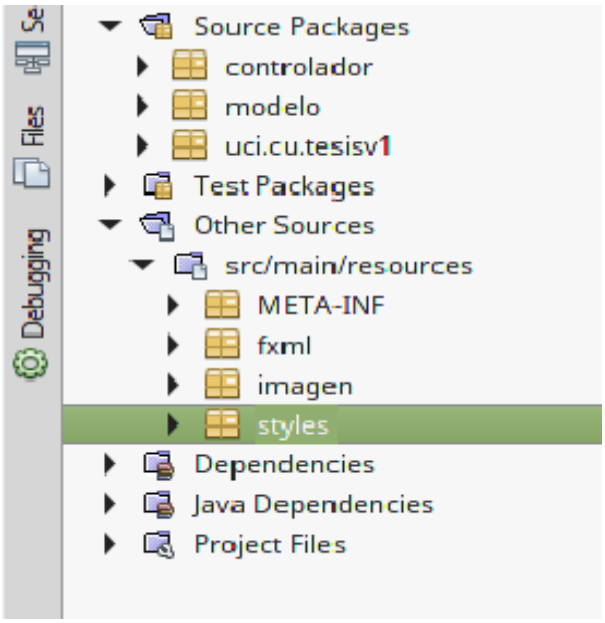

Figura 16. Paquetes de la aplicación.

3.2.2 Diagrama de componentes

Un componente es un bloque de construcción de software de cómputo. Según, la Especificación OMG del Lenguaje de Modelado Unificado define un componente como “una parte modular, desplegable y sustituible de un sistema, que incluye la implantación y expone un conjunto de interfaces” (Pressman, 2010).

Los componentes forman la arquitectura del software y, en consecuencia, juegan un papel en el logro de los objetivos y de los requerimientos del sistema que se va a construir. Como los componentes se encuentran en la arquitectura del software, deben comunicarse y colaborar con otros componentes y con entidades (otros sistemas, dispositivos, personas, etcétera) que existen fuera de las fronteras del software.

A continuación, se muestra el diagrama de componentes del módulo Ingresos, el cual es una representación gráfica del mismo relacionado con el resto de los componentes que intervienen en el SISEN, así como las dependencias entre estos. Se explican posteriormente las principales funcionalidades que ofrecen los componentes que se muestran en el diagrama.

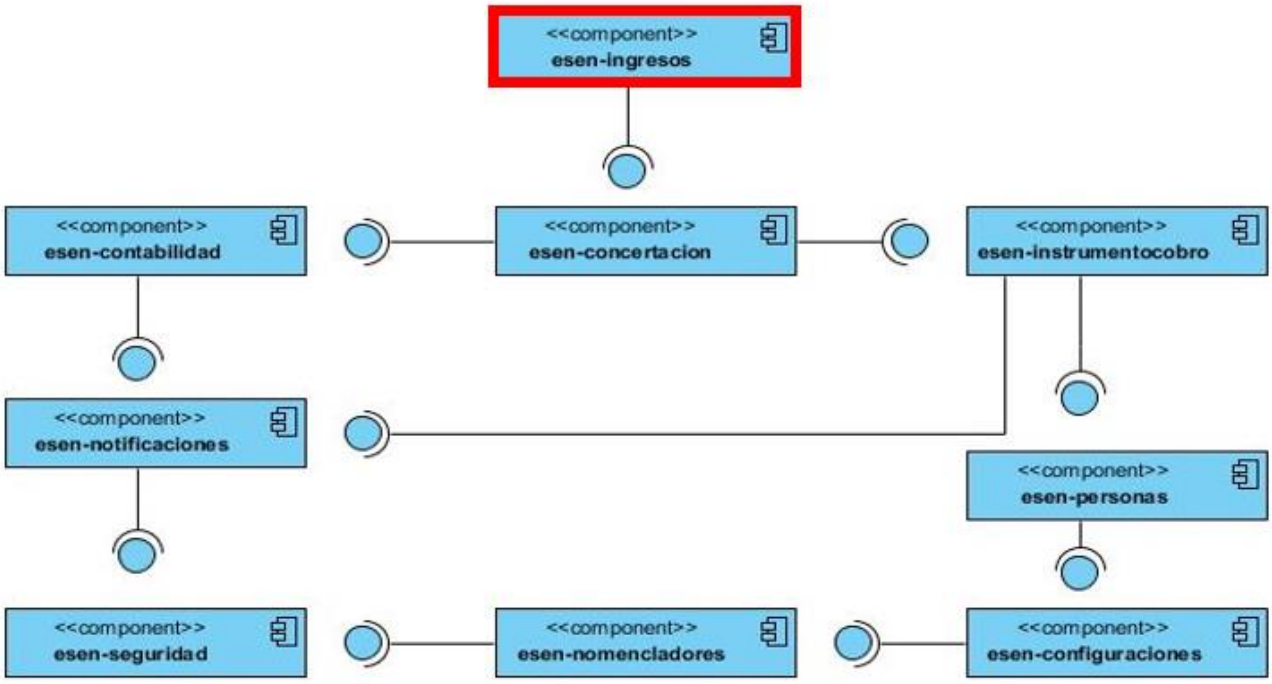

Figura 17. Diagrama de componentes.

Ingresos (esen-ingresos): Este componente resaltado en color rojo, el cual se desarrolla en el presente trabajo, permite el registro de los ingresos por conceptos de prima luego de su depósito en banco, de esta forma se encarga de emitir los registros contables establecidos, actualizar el estado de las pólizas pendientes de estos pagos, determinar las comisiones de los agentes involucrados y actualizar el estado de los instrumentos de cobro utilizados en el ingreso.

Concertación (esen-concertacion): Este se relaciona directamente con el componente Ingresos posibilitándole la captación de la información de las pólizas de los distintos productos de seguro que oferta la ESEN, haciendo cumplir una gran cantidad de reglas de negocio según las resoluciones y manuales de estos. Constituye uno de los componentes de negocio más importantes, pues da entrada al sistema a lo pactado entre la aseguradora y el asegurado.

Contabilidad (esen-contabilidad): Permite la gestión o importación del nomenclador de cuentas a utilizar en las operaciones contables del seguro de la ESEN en algunos bancos y en distintos formatos. El componente le permite al componente Ingresos la exportación de los comprobantes de operaciones contables relativos a todas las operaciones realizadas en el sistema para su importación en el sistema contable de la ESEN, así como la gestión de nomencladores contables como centros y unidades de costo, deudores, acreedores, sucursales y bancos. Además, posibilita la configuración de los comprobantes de operaciones contables y el registro manual de estos comprobantes a partir de los ingresos registrados en el sistema.

Instrumentos de cobro (esen-instrumentocobro): Posibilita el control de los instrumentos de cobro que utiliza la ESEN en el país, su registro en el sistema, su asignación, rehabilitación y cancelación dando cumplimiento a las reglas de negocio establecidas. También permite la emisión de información sobre estos mediante vistas y reportes que ofrecen sus detalles.

Personas (esen-personas): Permite la gestión de forma genérica para toda la aplicación de las personas, reconociendo en el caso del negocio de la ESEN, que tanto un asegurado, como un agente, tomador de una póliza, o tercero afectado entre otros, son finalmente personas, ya sean naturales o jurídicos. Esto permite de manera centralizada reutilizar y proporcionar información sobre estos cuando se estén gestionando en el sistema en distintos roles, ejemplo: un agente que sea a la vez un asegurado.

Notificaciones (esen-notificaciones): Permite la gestión de notificaciones entre usuarios, ya sea dirigidos a uno o varios usuarios específicos o a los que pertenecen a uno o varios roles a la vez.

Configuraciones (esen-configuraciones): Implementa funcionalidades que permiten la gestión de un grupo de configuraciones propias del negocio. Esto garantiza delegar sobre un único lugar el control de los parámetros que utiliza el componente Ingresos y otros más, durante su ejecución, favoreciendo buenas prácticas de desarrollo de software y ofreciendo al usuario en tiempo de ejecución, la posibilidad de modificación de estos parámetros sin tener que recurrir en muchos casos a la reimplementación.

Nomencladores (esen-nomencladores): Permite el trabajo con los nomencladores gestionables por los usuarios administradores del sistema durante su operación, así como de los fijos en base de datos

utilizados todos en el funcionamiento de los restantes componentes.

Seguridad (esen-seguridad): Desde este componente se pueden crear los roles con los permisos a las funcionalidades del sistema autorizadas, los usuarios del sistema asociados a estos roles, y la propia estructura del menú de la aplicación.

3.2.3 Diagrama de despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y la relación de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria (Pressman, 2010).

La figura 18 muestra el diagrama de despliegue del sistema. Posteriormente se describen las funciones de cada nodo que se muestra en el diagrama.

Figura 18. Diagrama de despliegue.

PC_Cliente: Computadora en la cual la aplicación se ejecutará a través de una aplicación de escritorio.

Base Datos SQLite: Es donde se encuentra la base de datos local que utiliza el sistema, este debe estar instalado en la misma computadora donde se encuentra el sistema.

3.3 Aplicación de las pruebas de software

3.3.1 Prueba de caja blanca

La técnica de caja blanca empleada en la solución desarrollada fue la Prueba del Camino Básico, la cual permite obtener una medida de la complejidad lógica del diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución, garantizando con estos que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

Para realizar esta técnica es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar. A continuación, se enumeran las sentencias de código del procedimiento realizado sobre el método *crearObjeto ()*, ver Figura 19, y el grafo de flujo asociado al mismo, ver Figura 20.

```java
@Override
public Ingreso[] crearObjeto(String[] valores) {
    int x = valores.length;
    Ingreso[] nuevosObjetos = new Ingreso[x];
    for (int i = 0; i < valores.length; i++) {
        String val = valores[i];
        String[] propiedades = val.split(",");
        Ingreso newIngreso = new Ingreso();
        newIngreso.setIdIngreso(Integer.parseInt(propiedades[0]));
        newIngreso.setFechaIngreso(LocalDate.parse(propiedades[1]));
        newIngreso.setImpIngreso(Double.parseDouble(propiedades[2]));
        newIngreso.setEstado(propiedades[3]);
        if (propiedades[4].equals("null")) {
            newIngreso.setComisionAgente(null);
        } else {
            newIngreso.setComisionAgente(Double.parseDouble(propiedades[4]));
        }
        if (propiedades[5].equals("null")) {
            newIngreso.setComisionBanco(null);
        } else {
            newIngreso.setComisionBanco(Double.parseDouble(propiedades[5]));
        }
        newIngreso.setDocBancoidDocBanco(Integer.parseInt(propiedades[6]));
        nuevosObjetos[i] = newIngreso;
    }
    return nuevosObjetos;
}
```

Figura 19. Método crearObjeto()

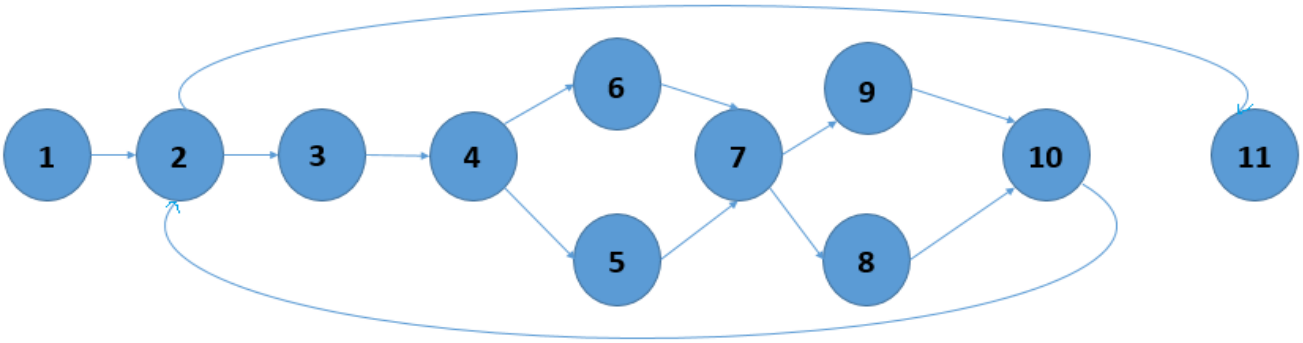

Figura 20. Grafo asociado al algoritmo del método crearObjeto().

El cálculo de la complejidad ciclomática es necesario efectuarlo mediante tres vías o fórmulas, se debe utilizar el mismo grafo en cada caso:

Fórmulas	Nomenclaturas	Resultados
(G) = (A – N) + 2	A: cantidad total de aristas del grafo. N: cantidad total de nodos del grafo.	(G) = (13 - 11) + 2 (G) = 4
(G) = P+1	P: cantidad total de nodos predicados. Un nodo es predicado cuando parten del mismo 2 o más aristas.	(G) = 3 + 1 (G) = 4
(G) = R	R: cantidad total de regiones existentes en el grafo, se incluye el área exterior del grafo, como una región más.	(G) = 4

Tabla 9. Fórmulas para el cálculo de la complejidad ciclomática.

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 4, lo que significa que existen 4 posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Número	Caminos básicos
1	1-2-3-4-5-7-8-10-2-11
2	1-2-3-4-5-7-9-10-2-11
3	1-2-3-4-6-7-8-10-2-11
4	1-2-3-4-6-7-9-10-2-11

Tabla 10. Caminos básicos del flujo.

Posteriormente de haber determinado los caminos básicos se procede a ejecutar los casos de pruebas para cada uno de estos. Para definir los casos de prueba es necesario tener en cuenta:

Descripción: Se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada y así

ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que serán la entrada al procedimiento.

Resultados esperados: Se expone el resultado que se espera que devuelva el procedimiento.

Resultados: Se muestra el resultado obtenido.

Seguidamente se presentan los casos de pruebas de caja blanca generados.

Camino básico #1: 1-2-3-4-5-7-8-10-2-11	
Descripción	Recibe un conjunto de valores y se crean los objetos de tipo ingreso
Condición de ejecución	Se debe tener los datos para crear el objeto ingreso
Entrada	String[ ] valores={“1,2014-03-23,10.50,Liquidado,null,null,45”}
Resultado esperado	Se crea un nuevo ingreso
Resultado	Satisfactorio

Tabla 11. Caso de prueba para el camino básico #1.

Camino básico #2: 1-2-3-4-5-7-9-10-2-11	
Descripción	Recibe un conjunto de valores y se crean los objetos de tipo ingreso
Condición de ejecución	Se debe tener los datos para crear el objeto ingreso
Entrada	String[ ] valores={“2,2014-04-23,30.00,En tránsito,null,20.50,45”}
Resultado esperado	Se construye un nuevo ingreso
Resultado	Satisfactorio

Tabla 12. Caso de prueba para el camino básico #2.

Camino básico #3: 1-2-3-4-6-7-8-10-2-11	
Descripción	Recibe un conjunto de valores y se crean los objetos de tipo ingreso

Condición de ejecución	Se debe tener los datos para crear el objeto ingreso
Entrada	String[ ] valores={“3,2014-04-20,50.00,En tránsito, 200.00,null ,44”}
Resultado esperado	Se construye un nuevo ingreso
Resultado	Satisfactorio

Tabla 13. Caso de prueba para el camino básico #3.

Camino básico #4: 1-2-3-4-6-7-9-10-2-11	
Descripción	Recibe un conjunto de valores y se crean los objetos de tipo ingreso
Condición de ejecución	Se debe tener los datos para crear el objeto ingreso
Entrada	String[ ] valores={“4,2014-05-25,100.00,Liquidado, 2.00,4.50 ,44”}
Resultado esperado	Se crea un nuevo ingreso
Resultado	Satisfactorio

Tabla 14. Caso de prueba para el camino básico #4.

3.3.2 Prueba de caja negra

Las pruebas se realizaron por el método partición de equivalencia sobre todos los requisitos funcionales, haciendo uso de los casos de prueba correspondientes a cada uno de estos. A continuación, se muestra un ejemplo:

3.3.3 Caso de prueba para el requisito Registrar ingreso

Condiciones de ejecución

1. Se debe identificar y autenticar en el sistema y contar con los permisos necesarios para ejecutar esta acción.
2. Selecciona la opción Gestionar ingresos/ Registrar ingreso.

Nombre del requisito	**Descripción general**	**Escenarios de pruebas**	**Flujos del escenario**
1:Registrar	El objetivo de este	EP 1.1: Adicionar	1. Se presiona el botón Adicionar. 2. Se muestra la interfaz con los

ingresos.	requisito es registrar un ingreso por póliza.	ingreso.	campos necesarios para registrar un ingreso en el sistema. 3. Se introducen los datos correctamente. 4. Se presiona el botón Aceptar. 5. Se muestra un mensaje indicando que se ha registrado correctamente el ingreso. 6. Se presiona el botón Aceptar de la ventana de información.
		EP 1.2: Adicionar un ingreso con datos obligatorios vacíos.	1. Se presiona el botón Adicionar para registrar el ingreso. 2. Se dejan campos obligatorios vacíos. 3. El sistema muestra los campos vacíos y el mensaje: “Existen campos obligatorios vacíos.”
		EP 1.3: Cancelar	1. Se presiona el botón Adicionar. 2. Se muestra la interfaz Adicionar ingreso. 3. Se introducen o no los datos del ingreso. 4. Se presiona el botón Cancelar. 5. Se limpian los campos.

Tabla 15. Diseño de caso de prueba del requisito Registrar ingresos.

Id del escenario	**Escenario**	**Descripción**	**Respuesta del sistema**
EP 1.1	Adicionar un ingreso entrando datos válidos	Se llenan los campos con datos válidos	• El sistema muestra una ventana de información con el texto: “El ingreso se ha registrado correctamente.” • Se acepta el mensaje de información y se limpian todos los campos.

EP 1.2	Adicionar un ingreso entrando datos inválidos	Se llenan los campos con datos inválidos	• El sistema no permite introducir datos inválidos. Cada campo tiene su restricción de entradas de datos evitando datos incorrectos.

Tabla 16. Juego de datos a probar del requisito Registrar ingresos.

3.3.4 Resultados de las pruebas aplicadas al componente

Como resultado de la aplicación de pruebas de caja blanca al sistema fueron analizadas las funcionalidades del mismo y no se detectan errores. Esto permite afirmar que el flujo de trabajo de las funciones es correcto.

Para la aplicación de la prueba de caja negra se realizaron 5 diseños de casos de prueba, los cuales se basan en una evaluación de las clases de equivalencia para una condición de entrada. Como resultado de la aplicación de las pruebas de caja negra fueron identificadas un total de 19 No Conformidades (NC), las cuales fueron corregidas a lo largo de dos iteraciones. La relación de NC y la clasificación de las mismas se representan de la siguiente forma:

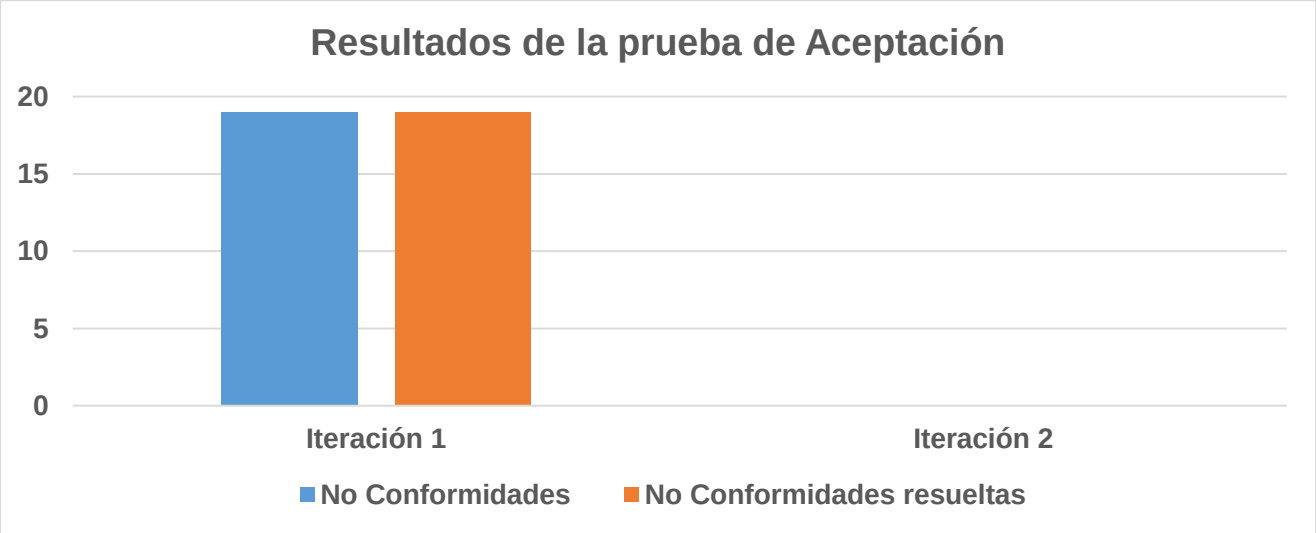

Figura 21. Resultados de la prueba de Aceptación. Método de prueba de Caja Negra.

3.4 Validación de la investigación

Para validar la investigación y resolver la necesidad de reducir el tiempo en la gestión de los ingresos de la ESEN, se efectuó una entrevista a una agente coordinadora de dicha empresa, evaluando cuánto demora realizar el registro de un ingreso manualmente. Luego de comprobar el proceso de registro de

un ingreso, las comisiones de agentes y sus respectivos ajustes de comisión, resultó un tiempo de 12 minutos aproximadamente.

En cuanto a la aplicación desarrollada, se empleó el enfoque de la norma ISO 9241-11 para medir usabilidad. Esta define la medida en que un producto puede ser utilizado por usuarios especificados para alcanzar objetivos específicos con eficacia, eficiencia y satisfacción en un contexto determinado de uso. (Mifsud, 2015)

La métrica ISO / IEC 9126-4 recomienda que las métricas de usabilidad incluyan:

- Eficacia: La precisión y la integridad con la que los usuarios alcanzan los objetivos especificados.
- Eficiencia: Los recursos gastados en relación con la exactitud y la integridad con la que los usuarios alcanzan los objetivos.
- Satisfacción: La comodidad y la aceptabilidad del uso.

La eficiencia se mide en términos de tiempo de tarea, es decir, el tiempo (en segundos y / o minutos) que el usuario toma para completar una tarea con éxito. El tiempo que se tarda en completar una tarea se puede calcular simplemente restando el tiempo de inicio del tiempo de finalización, como se muestra en la siguiente ecuación: Tiempo de tarea = Hora de finalización - Hora de inicio.
Existen dos formas de medir la eficiencia:

1. Eficiencia basada en el tiempo.
2. Eficiencia relativa global.

Para conocer cuánto demora el registro de un ingreso en la aplicación de escritorio desarrollada se utilizará la primera forma.

Eficiencia basada en el tiempo

$$Time\ Based\ Efficiency = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}}}{NR}$$

N = El número total de tareas (metas).
R = El número de usuarios.
Nij = El resultado de la tarea i por el usuario j. Si el usuario completa con éxito la tarea, entonces Nij = 1, si no, entonces Nij = 0.
Tij = Tiempo empleado por el usuario j para completar la tarea i. Si la tarea no se completa correctamente, el tiempo se mide hasta el momento en que el usuario abandona la tarea.

Cálculo de la métrica

N = El número total de tareas = 3 (tareas correspondientes a registrar un ingreso, una comisión adicional y un ajuste de comisión de un agente)
R = El número de usuario = 4
Usuario 1: Nij = 3 y Tij = 80 segundos
Usuario 2: Nij = 3 y Tij = 85 segundos
Usuario 3: Nij = 0 y Tij = 100 segundos
Usuario 4: Nij = 3 y Tij = 79 segundos

Colocando los valores anteriores en la ecuación:
TBE = (3/80 + 3/85 + 0/100 + 3/79) /3*4 = 0.009 segundos/tareas
3 tareas / 0.009 tareas/segundos = 333 segundos = 5 minutos aproximadamente
Luego de realizado el análisis de la métrica basada en el tiempo se obtiene como resultado 0.009 tareas/segundos. Esto significa que un usuario demora 5 minutos aproximadamente en realizar una tarea, lo cual evidencia que el sistema desarrollado reduce el tiempo al registrar un ingreso.

3.5 Conclusiones parciales del capítulo

Luego de haberle dado cumplimiento a los objetivos trazados para el tercer capítulo, se arribó a las siguientes conclusiones:

- Se implementó los requerimientos funcionales en correspondencia a sus descripciones, permitiendo obtener una primera versión funcional del producto en la variante de escritorio.
- Los métodos de prueba caja blanca y caja negra permitieron validar el correcto funcionamiento del sistema. Como resultado se obtuvo 19 no conformidades, teniendo en cuenta los 5 diseños de casos de pruebas analizados, estas fueron resueltas inmediatamente después de detectadas, mientras que en el código no se detectaron errores.
- La validación de la investigación a través de la entrevista a un agente de la ESEN y el uso del enfoque ISO / IEC 9126-4 para las métricas de usabilidad, demostró que con el sistema desarrollado el tiempo de duración de la gestión de un ingreso es de 5 minutos, lo cual evidencia que minimizó el tiempo con respecto a los 12 minutos que demora realizar la gestión manual.

Conclusiones generales

Durante el desarrollo de la investigación se planteó la necesidad de diseñar e implementar un módulo para gestionar los ingresos del Sistema Integral de Seguros Nacionales, dándole así cumplimiento al objetivo propuesto de la siguiente forma:

- El estudio de los aspectos teóricos relacionados con la gestión de almacenamiento de información fuera de línea permitió identificar las características principales del componente, propiciando un mejor entendimiento del diseño de la propuesta de solución.
- El análisis de los 8 requisitos funcionales y los 10 requerimientos no funcionales permitieron adaptar y diseñar interfaces según las necesidades del cliente, posibilitando una mejor guía para la implementación del componente.
- La implementación permitió un componente de acuerdo a los requisitos identificados por el cliente, logrando una primera versión funcional del producto en la variante de escritorio para el SISEN.
- Con los métodos de prueba caja blanca y caja negra se lograron detectar 19 no conformidades, teniendo en cuenta los 5 diseños de casos de pruebas, las cuales fueron resueltas inmediatamente después de detectadas a lo largo de dos iteraciones.
- El enfoque de la norma ISO 9241-11 para las métricas de usabilidad, permitió validar la investigación y arrojó como resultado que el sistema desarrollado minimiza el tiempo de la gestión de los ingresos en aproximadamente 5 minutos en comparación con la gestión manual.

Por lo anteriormente descrito se determinó que el objetivo general de la investigación: desarrollar el componente Ingresos para la variante de escritorio del Sistema Integral de Seguros Nacionales, cumpliendo con los requisitos identificados, de manera que se registre toda la información del proceso de ingresos, comisiones adicionales y ajustes de comisión de agentes, cuando esté sin servicio el servidor central, fue cumplido en su totalidad.

Recomendaciones

En función del constante proceso de mejora y evolución que es inherente a todo sistema de software se recomienda lo siguiente:

- Desarrollar una variante de escritorio para los demás módulos del SISEN.
- Realizar la sincronización de la aplicación desarrollada con el SISEN para que el componente obtenga una información actualizada de los ingresos.

Bibliografía

Calleja, Manuel Arias. 2017. Centro de Investigación sobre Sistemas Inteligentes de Ayuda a la Decisión. CISIAD. [En línea] 2017. www.cisiad.uned.es/carmen/estilo-codificacion.pdf.
Erika Camacho, Fabio Cardeso y Gabriel Nuñez. 2014. Arquitecturas de Software. 2014.
ESEN. 2016. Empresa de Seguros Nacionales. [En línea] 2016. http://www.esen.cu/index.php?page=asegurarse.
Fowler, Martin. 2003. Patterns of Enterprise Application Architecture. 2003.
Gauchat, Juann Diego. 2012. El gran libro de HTLM5,CCS3 y Javascript. Barcelona : s.n., 2012.
González, Rolando Alfredo Hernández León y Sayda Coello. 2012. EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN. Ciudad Habana : s.n., 2012.
Hibernate.org. 2017. Hibernate. [En línea] 2017. http://hibernate.org/orm/.
Hugo Michael Marca Huallpara, Nancy Susana Quisbert. 2016. Análisis y diseño de sistemas(Trabajo de investigación y exposición).´´Diagrama de Despliegue´´. 2016.
Lancker, Luc Van. 2013. Los API JavaScript de HTML5: Integre la potencia de HTML5 en sus aplicaciones web. Barcelona : s.n., 2013.
Larman, Craig. 2006. Parte I « El Mundo Informático, Patrones Grasp. UML y Patrones. 2006.
Lowe, Doug. 2014. JavaFX For Dummies. JavaFX For Dummies. 2014.
Mahemoff, Michael. 2013. "Offline": What does it mean and why should I care? HTML5 Rocks. [En línea] 29 de 10 de 2013.
Mark Lorenz, Jeff Kidd. 1994. Object-Oriented Software MetricsObject-Oriented Software. 1994.
Navathe, Ramez Elmasri y Shamkant B. 2015. Fundamentos de sistemas de bases de datos 5ta edicion. [En línea] 21 de 07 de 2015. http://gunuweb.net/libros/base-de-datos/fundamentos-de-sistemas-de-bases-de-datos-5ta-edicion-ramez-elmasri-shamkant-b-navathe/.
netbeans.org. NetBeans. [En línea] www.netbeans.org.
Rodríguez, J. 2014. Patrones de diseño y frameworks. 2014.
Sayas, Lorente. 2013. La eficacia de la integración offline y online en la estrategia de comunicación corporativa. [En línea] 2013. Memoria.pdf.
Solis, Javier. 2012. CIO América Latina. Gestión y almacenamiento eficiente de la información . [En línea] Brand Manager Enterprise Storage Solutions Dell Latin America, 11 de 12 de 2012. Gestión y almacenamiento eficiente de la información para Latinoamérica - CIOAL The Standard IT.htm. SQLite.La Base de Datos Embebida. **Rómmel, Filein. 2017.** 2017.
sqlite.org. SQLite. [En línea] http://www.sqlite.org/.
SQLiteExpert. 2017. SQLiteExpert database administration. [En línea] 2017. http://www.sqliteexpert.com/.
Universidad de las Ciencias Informáticas. 2015. Metodología de desarrollo para la Actividad productiva de la UCI. 2015.
Valdeande, María de los Ángeles Hortiguela. 2012. Análisis y gestión de los instrumentos de cobro y pago. s.l. : S.A. EDICIONES PARANINFO, 2012.
Villarrubia, Celia. 2013. Datacenter Dynamics. [En línea] 10 de Deciembre de 2013. http://www.datacenterdynamics.es/focus/archive/2013/12/casi-8000-d%C3%B3lares-de-p%C3%A9rdidas-por-minuto-de-inactividad-en-el-cpd.
Visual Paradigm. 2017. software.com.ar. [En línea] Targetware, 2017. http://www.software.com.ar/p/visual-paradigm-para-uml#product-description.

Referencias

Castro, Raúl. 2009. Esicuba. [En línea] 2009. http://www.esicuba.cu/Documentos/Decreto263delContratodeSeguro.pdf..
—. 2010. Gaceta Oficial de la República de Cuba. 2010.
Cobas, Ing. Leydis Castellanos. 2015. CEIGE_ESEN_Especificación_de_requisitos_de_software. 2015.
—. 2015. CEIGE_ESEN_Modelo_conceptual. 2015.
Erika Camacho, Fabio Cardeso y Gabriel Nuñez. 2014. Arquitecturas de Software. 2014.
ESEN. 2016. Empresa de Seguros Nacionales. [En línea] 2016. http://www.esen.cu/index.php?page=asegurarse.
Fowler, Martin. 2003. Patterns of Enterprise Application Architecture. 2003.
Gauchat, Juann Diego. 2012. El gran libro de HTLM5,CCS3 y Javascript. Barcelona : s.n., 2012.
González, Rolando Alfredo Hernández León y Sayda Coello. 2012. EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN. Ciudad Habana : s.n., 2012.
Lancker, Luc Van. 2013. Los API JavaScript de HTML5: Integre la potencia de HTML5 en sus aplicaciones web. Barcelona : s.n., 2013.
Larman, Craig. 2006. Parte I « El Mundo Informático, Patrones Grasp. UML y Patrones. 2006.
Leandro Alegsa. 2016. Diccionario de informática y tecnología. [En línea] 27 de 06 de 2016.
Lowe, Doug. 2014. JavaFX For Dummies. JavaFX For Dummies. 2014.
Naranjo, Leonardo Darell Antúnez. 2012. Implementación del Sub-sistema de gestión de información personal de la Facultad 3. La Habana : Universidad de las Ciencias Informáticas, 2012.
Navathe, Ramez Elmasri y Shamkant B. 2015. Fundamentos de sistemas de bases de datos 5ta edicion. [En línea] 21 de 07 de 2015. http://gunuweb.net/libros/base-de-datos/fundamentos-de-sistemas-de-bases-de-datos-5ta-edicion-ramez-elmasri-shamkant-b-navathe/.
netbeans.org. NetBeans. [En línea] www.netbeans.org.
Pressman, Roger. 2010. Ingeniería de software enfoque práctico.7edición. Ingenieria de software enfoque practico. 2010.
Real Academia Española. 2014. DLE : váucher. Diccionario de la lengua española.Edición del Tricentenario. [En línea] 2014. http://dle.rae.es/?id=bPiXazx.
Rodríguez, J. 2014. Patrones de diseño y frameworks. 2014.
Sánchez, Carlos Martín. 2014. Vin'Blogs. [En línea] 07 de 06 de 2014. http://carlosvin.github.io/en/posts/java-embedded-db-performance-comparison.html.
Sayas, Lorente. 2013. La eficacia de la integración offline y online en la estrategia de comunicación corporativa. [En línea] 2013. Memoria.pdf.
Solis, Javier. 2012. CIO América Latina. Gestión y almacenamiento eficiente de la información . [En línea] Brand Manager Enterprise Storage Solutions Dell Latin America, 11 de 12 de 2012. Gestión y almacenamiento eficiente de la información para Latinoamérica - CIOAL The Standard IT.htm.
Sommerville, Ian. 2005. Ingeniería del Software 7ma. Ed parte 2. Ingeniería del Software. 2005. SQLite.La Base de Datos Embebida. **Rómmel, Filein. 2017.** 2017.
Universidad de las Ciencias Informáticas. 2015. Metodología de desarrollo para la Actividad productiva de la UCI. 2015.
Valdeande, María de los Ángeles Hortiguela. 2012. Análisis y gestión de los instrumentos de cobro y pago. s.l. : S.A. EDICIONES PARANINFO, 2012.
Villarrubia, Celia. 2013. Datacenter Dynamics. [En línea] 10 de Deciembre de 2013. http://www.datacenterdynamics.es/focus/archive/2013/12/casi-8000-d%C3%B3lares-de-p%C3%A9rdidas-por-minuto-de-inactividad-en-el-cpd.