Universidad de las Ciencias Informáticas.

Facultad 3

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Diseño e implementación de las mejoras funcionales para el subsistema Finanzas en su versión 1.1.

Autor: José Luis Rodríguez Morejón

Tutor: Ing. Iyugnis Leyva Báez

Co-tutor: Ing. Tania Teresa Loureiro Valladares

Ciudad de la Habana, 21 de junio de 2013.

"Año 55 de la Revolución"

Declaración de autoría.

Declaro ser único autor del presente trabajo de tesis y se le reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma el presente a los ____ días del mes de _____ del año 2013.

_____

José Luis Rodríguez Morejón

Autor

_____

Ing. Iyugnis Leyva Báez

Tutor

_____

Ing. Tania Teresa Loureiro Valladares

Co-tutor

Nuestra revolución está en marcha; y la utilidad de lo que se dice hoy, no se verá hasta mañana. Decir es hacer, cuando se dice a tiempo.

José Martí

DEDICATORIA

Dedico todo el trabajo, esfuerzo y estos cinco años que me convirtieron en Ingeniero en Ciencias de la Informática a mis abuelos paternos Julia y Luis, que no pudieron verme ingeniero pero fueron sus sueños, aquí se los hago realidad.

AGRADECIMIENTOS

A mi madre y padrastro que sin ellos no hubiera sido la mitad de lo que soy hoy.

A mi padre por preocuparse y ocuparse tanto.

A mi tía Noelvia por toda su ayuda.

A mi tío Papito por correr tanto conmigo en todos mis viajes.

A toda mi familia, toda, que estaban pendiente estos cinco años

A mis tutoras Tania e Iyugnis porque sin ellas ahora mismo no pudiera estar celebrando, a ellas les debo esta.

A todo mi tribunal y oponente, por darme tanto apoyo y un voto de confianza desde el primer encuentro.

A mi compañera de estudios Katia que siempre estuvo dispuesta a toda hora en ayudarme en lo que fuese necesario, a ella le debo gran parte de esta tesis.

A mi amigo Adrián, por aguantarme todos este tiempo.

A Nosdaly por preocuparse constantemente por mi situación.

A mis amistades, a todos los que de una forma u otra contribuyeron en estos cinco años de mi carrera.

RESUMEN

Para mejorar los resultados productivos y alcanzar una planificación de los recursos materiales y financieros, en Cuba se emplean algunos Sistemas de Planificación de Recursos Empresariales (ERP, del inglés). Sin embargo, los mismos no cumplen con la totalidad de los requisitos funcionales y la independencia tecnológica que se necesita. Por tal motivo se desarrolla en la Universidad de las Ciencias Informática (UCI) de conjunto con otras entidades el Sistema de Planificación de Recursos Empresariales Cedrux, integrado por un conjunto de subsistemas que responden a los diferentes procesos de negocios económicos de las empresas. En el área que integra le gestión de actividades financieras se necesita de algunas funcionalidades con las que no se cuenta, además de algunas mejoras requeridas por las empresas.

El objetivo fundamental del presente trabajo consiste en realizar el diseño e implementación de una nueva versión que solucione las deficiencias detectadas en las pruebas de certificación realizadas por especialistas funcionales de las empresas consultoras CONAVANA, Interaudit y CANEC. El desarrollo de la investigación está regido por el Modelo de Desarrollo establecido por el Centro de Informatización de Gestión de Entidades (CEIGE). La implementación se realiza en el marco de trabajo Sauxe, el desarrollo del mismo está basado en otros marcos de trabajo como ExtJS para la capa de presentación, Zend Framework para el manejo de la lógica de negocio y Doctrine para el acceso a datos. Se pretende obtener un producto totalmente funcional que cumpla con las necesidades del cliente, permitiendo así que mejore el funcionamiento en general del subsistema Finanzas perteneciente a Cedrux.

Palabras claves: Cedrux, Sistema de Gestión Empresarial, Gestión financiera.

TABLA DE CONTENIDO

INTRODUCCIÓN

En el afán por incrementar la productividad, eficiencia y rendimiento de las empresas, se han buscado vías para mejorar la gestión económica empresarial. Actualmente existe gran flujo de datos que requieren un alto consumo de tiempo y recursos para su manejo. Las empresas requieren de sistemas que les brinden control y acceso a toda la información de forma confiable, precisa y oportuna. Por tal motivo, surgen los Sistemas de Planificación de Recursos Empresariales (*ERP*, *Enterprise Resource Planning*), para lograr alcanzar la optimización de procesos, como eslabón fundamental de la integración de la información en las diferentes áreas o departamentos de las entidades. El desarrollo de la economía cubana está vinculado en la actualidad a los avances relacionados con las nuevas tecnologías de la información y las comunicaciones, insertarse en ese mercado cada día más, es uno de los objetivos primordiales para la mejora de la economía.

En el presente, existen varios paquetes de software que facilitan los procesos en las empresas cubanas. Ejemplo de ello se tiene VERSAT-Sarasola, RODAS XXI, SISCONT5, CONDOR, ASSETS-NS, Openbravo, entre otros; pero no se ha logrado tener una herramienta estándar que incluya funcionalidades afines a las particularidades de los procesos económicos del país. A partir de esta óptica surge la idea de desarrollar un sistema de gestión empresarial acorde a las necesidades reales de las empresas cubanas y aplicables a cualquier empresa productora de bienes o servicios. La Universidad de las Ciencias Informáticas, de conjunto con otras entidades desarrolla el Sistema Integral de Gestión Empresarial Cedrux, con el cual propone crear un sistema de gestión empresarial nacional, que use software libre, contribuya a la independencia tecnológica, permita mejorar los resultados económicos en el país y logre una eficiente planificación de los recursos materiales y financieros.

El ERP cubano tiene entre sus objetivos integrar la información financiera de una organización, unificando el juego de datos monetarios de esa empresa con el de las diferentes unidades comerciales que interactúen con el software. La integración financiera de Cedrux se gestiona mediante el subsistema Finanzas, en la que están contenidos los módulos: Cobros y Pagos (COPA), Banco y Caja. Es importante señalar que la solución Finanzas realiza las operaciones financieras considerando la dualidad monetaria existente en Cuba, permitiendo a las entidades realizar operaciones en distintas monedas.

En las pruebas de certificación realizadas por especialistas funcionales de las empresas consultoras CONAVANA, Interaudit y CANEC, fueron identificadas un conjunto de no conformidades y pedidos de cambio en todos los módulos del subsistema, tales como: Actualmente el módulo Banco presenta problemas en la cancelación directa de un instrumento bancario, en la visualización de los comprobantes generados y en la adición de estados de cuentas. Por su parte el módulo Caja al iniciarse un nuevo ejercicio económico no tenía en cuenta una nueva numeración al adicionar un nuevo talonario, al liquidar un anticipo de dieta de gastos de viajes, se tenían unidas la funcionalidades liquidar y contabilizar, y no tenía concebida las funcionalidades modificar y confirmar, también el sistema no tenía concebido el control de firmas autorizadas, así como la modificación y eliminación de una apertura de fondo registrada. En el módulo COPA se señaló que no se podía realizar la liquidación parcial de un pago anticipado, la cancelación por expediente de proyecto no estaba aún terminada en el componente Derecho u obligación. También el sistema no mostraba el listado de los cobros y pagos anticipados pendientes ni se realizaba el cambio de cuenta en los documentos derecho de cobro y obligación de pago, por lo que se adicionó el componente Cambio de cuenta con los requisitos de buscar el documento a cambiar de cuenta, además se debe seleccionar la operación contable que está asociada a la nueva cuenta que se le va a asociar al documento.

A partir de lo expuesto, se define como **problema a resolver** que la gestión de las finanzas en Cedrux se ve afectada al no satisfacer los requisitos funcionales de la versión 1.1 del subsistema Finanzas.

En este contexto se tiene como **objeto de estudio** las soluciones informáticas para la gestión de las finanzas, centrando como **campo de acción** el diseño e implementación de la versión 1.1 del subsistema finanzas.

Se plantea como **objetivo general**: Realizar el diseño y la implementación de la versión 1.1 del subsistema Finanzas que posibilite satisfacer sus requisitos funcionales mejorando la gestión de las finanzas en Cedrux.

Para dar solución al objetivo propuesto se desglosan los siguientes **objetivos específicos:**

- ❖ Fundamentar la investigación mediante la elaboración del Marco Teórico para sustentar los conceptos y la propuesta de desarrollo de la solución.

- ❖ Realizar el diseño de la propuesta de solución para obtener el punto de partida de la implementación.
- ❖ Realizar la implementación para satisfacer los requisitos funcionales.
- ❖ Realizar pruebas de caja negra para validar la solución.

Teniendo como **idea a defender** que la realización del diseño y la implementación de la versión 1.1 del subsistema Finanzas posibilitarán satisfacer sus requisitos funcionales mejorando la gestión de las finanzas en Cedrux.

Con el propósito de cumplir los objetivos planteados con el desarrollo de la investigación se utilizaron los **métodos de investigación** siguientes:

De los métodos teóricos se utilizaron:

❖ **Histórico – Lógico:** Para estudiar toda la trayectoria y tendencias actuales en el desarrollo de los procesos financieros y determinar su influencia en el problema actual de la investigación. Las etapas más significativas de su desarrollo y sus conexiones históricas fundamentales de forma cronológica y lógica.

❖ **Analítico – Sintético:** Para realizar un análisis de la información empleada para la investigación, así como las especificaciones de los diferentes procesos financieros y centrándolos en la problemática; y una valoración de las diferentes características de los diferentes procesos financieros y poder obtener la solución centrada en los objetivos de la investigación.

Dentro de los métodos empíricos se utilizaron:

❖ **Observación:** La misma consiste en un estudio del proceso financiero mediante la percepción y registro planificado y sistemático de su comportamiento. Para determinar cómo quedaría el nuevo desarrollo y su influencia en la integración con el actual sistema.

El trabajo de diploma cuenta con Introducción, tres Capítulos, Conclusiones, Recomendaciones, Bibliografía y Glosario de Términos.

Capítulo 1: “Fundamentación Teórica”. Incluye un estudio de los sistemas informáticos financieros más significativos en nuestro país, se describen las herramientas y tecnologías a utilizar así como los lenguajes de programación. Se realiza un estudio del patrón arquitectónico modelo vista controlador así como de los patrones de diseño.

Capítulo 2: “Propuesta de solución”. Se describe la solución para la mejora de las propuestas funcionales del subsistema Finanzas v1.1 de Cedrux.

Capítulo 3: “Validación de la Solución”. Se evalúa el diseño propuesto a partir de las métricas de diseño TOC y RC. Se valida la implementación a partir de las pruebas de caja negra realizándose una valoración de la misma.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se estudiarán los principales ERP utilizados en Cuba, prestando atención a la gestión financiera en los mismos. Se describen las herramientas y tecnologías a utilizar así como los lenguajes de programación teniendo en cuenta el marco de trabajo Sauxe y el modelo de desarrollo definido por el CEIGE. Se hace referencia a los patrones de diseño y arquitectónico, además del estudio sobre las métricas TOC y RC para validar el diseño.

1.2 Sistemas de Planificación de recursos empresariales

Un sistema ERP es un conjunto de programas integrados que apoya las principales actividades organizacionales tales como producción, logística, finanzas, contabilidad, ventas y recursos humanos. Esto significa que lo que se trata, es de contar con un solo programa de software que satisfaga las necesidades de todos los departamentos de la empresa. (Jiménez, y otros, 2010)

Existen varios software para la gestión de los recursos empresariales como son SAP, VERSAT-Sarasola, ASSET NS, etc. Estos productos en su mayoría ofrecen adaptabilidad a los cambios de negocio, logran una perfecta integración de los procesos y proporcionan opciones de seguridad que le permiten limitar el acceso a los diferentes procesos del sistema de acuerdo con el perfil de cada usuario. Entre sus características fundamentales se encuentran en el caso de SAP, que ayuda a la toma de decisiones pues se accede a la información indicada en tiempo real para identificar los inconvenientes en forma temprana e ir en busca de las oportunidades proactivamente. En VERSAT-Sarasola están recogidas casi la totalidad de las operaciones financieras que pueden generarse en cualquier entidad y ASSET NS es un sistema flexible, amigable, con ayuda en línea que puede ser instalado en una microcomputadora o sobre varias, funcionando en ambiente multiusuario incluidas estaciones remotas.

El estudio de estos sistemas no arrojaron un resultado directo a la solución de la presente investigación pero resultó un punto de partida para comprender desde otra óptica, el funcionamiento de los procesos empresariales y específicamente los financieros.

1.3 Sistema Integral de Gestión Cedrux

El Sistema Integral de Gestión Cedrux es desarrollado por la UCI en coordinación con las entidades DESOFT y TEICO de Villa Clara y Ministerio de Finanzas y Precios. Surge con el objetivo de satisfacer todos los requerimientos funcionales y técnicos que estén a la altura de las mejores soluciones de este tipo a nivel mundial. La aplicación debe vencer las barreras de software existentes para migrar hacia las nuevas posibilidades que brinda el Software Libre. De manera general, el sistema constituye una solución completa a todas las áreas de las entidades que permite optimizar los procesos empresariales garantizando la eficiencia organizacional del país. (Arias, y otros, 2010)

Dentro de las principales características de Cedrux se encuentra su independencia tecnológica. Funciona sobre plataforma web y puede mantener interoperabilidad con otros sistemas. Es multimoneda, multientidad y como característica particular de Cuba permite hacer las operaciones sobre la doble moneda (peso cubano y peso convertible). Mantiene el control de fechas, es transaccional, con integridad funcional y posibilita el tratamiento estadístico del procesamiento de la información. El sistema Cedrux establece elementos arquitectónicos aplicando el estilo de Arquitectura basada en componentes siguiendo los principios de la Arquitectura Orientada a Servicios (*SOA, Service-oriented architecture*) y a través del patrón Modelo-Vista-Controlador. (Manfugás, 2011) El ERP está dividido en varios subsistemas, entre los que se encuentran:

Contabilidad: El subsistema Contabilidad es el encargado de llevar el control de todos los movimientos u operaciones contables que se realizan en cualquier entidad, por ejemplo: la apertura de una cuenta y la creación y recepción de comprobantes. Se creó para organizar y gestionar todo lo relacionado con la contabilidad acorde a las normativas del país con el objetivo de mejorar el control y el seguimiento de las operaciones contables en las empresas cubanas. (Polanco, 2010)

Costos y procesos: Este subsistema fue creado con la misión fundamental de gestionar los costos y procesos conforme a las normativas establecidas en Cuba con el propósito de optimizar el control y el seguimiento de los gastos en las empresas cubanas. La tarea esencial del subsistema es llevar el control de todos los movimientos u operaciones de costos que se realicen en cualquier entidad, por ejemplo: las operaciones para ajustar los costos de los productos en producción terminada. (Polanco, 2010)

Estructura y composición: El subsistema Estructura y Composición permite definir el entorno estructural organizativo en el cual se van a ubicar las entidades (que es la estructura superior o externa) y la estructura interna dentro de cada una de las mismas; además de definir los cargos y puestos de trabajos por los cuales estará compuesta cada área de una unidad. También se puede especificar el nivel jerárquico que existe entre las entidades que se estén gestionando, es decir, a quien se subordina las mismas. (Rivas, 2010)

Configuración: El subsistema de Configuración permite gestionar toda la configuración que se establece en una entidad; brinda facilidades de adaptación que guiarán a las entidades en el manejo de sus procesos contables, financieros, logísticos, entre otros procesos y funciones administrativas, permitiendo de este modo, adaptar el sistema a sus peculiaridades y entorno de trabajo. (Rivas, 2010)

Multimoneda: El subsistema Multimoneda permite que las entidades definan y empleen diversas monedas, según lo requieran, en la ejecución de transacciones u operaciones económicas y su registro. Igualmente posibilita la gestión de las tasas de cambios y la reevaluación de cuentas, entre otras funcionalidades, que son esenciales para llevar el control de la gestión económica y del estado financiero en cualquier entidad.

1.3.1 Subsistema Finanzas

El área de Finanzas en los ERP tiene como principal objetivo el uso óptimo de recursos, en cuanto a cantidad, calidad y oportunidad, tanto de las fuentes que suministran los fondos como del empleo que de ellos se hacen. Esta se encarga de la obtención de fondos y del suministro del capital que se utiliza en el funcionamiento de la empresa, procurando disponer de los medios económicos necesarios para cada uno de los departamentos, logrando que puedan funcionar debidamente y teniendo implícito el máximo aprovechamiento y administración de los recursos financieros. (Jiménez, y otros, 2010) El subsistema Finanzas cuenta con tres módulos fundamentales Caja, COPA y Banco.

Caja: Su función fundamental consiste en controlar los fondos monetarios existentes en una caja. Estos fondos comprenden el efectivo pendiente de depositar por cobros efectuados, así como cualquier otro medio monetario en poder de la entidad. Una caja está asociada a una cuenta y su fondo puede ser de diferentes tipos, lo que condiciona las operaciones a realizar en caja. En este

subsistema son tratados además los anticipos otorgados para viajes, contemplando las particularidades establecidas en caso de que el anticipo sea para viaje nacional o extranjero. También se tendrá en cuenta el modo de pago si es en efectivo o por cheque. (Bailly, 2010)

El módulo Caja posee un total de 20 funcionalidades, agrupadas en los componentes Configuración, Carga inicial, Movimientos, Arqueo, Cierre y Recuperaciones. Las funcionalidades a modificar partiendo de las no conformidades y pedidos de cambios se concentran en los componentes Configuración y Movimientos. A continuación se presenta una descripción de las funcionalidades:

Gestionar talonario

Se detectó que el sistema al iniciarse un nuevo ejercicio económico no tiene en cuenta una nueva numeración al adicionar un nuevo talonario, debiendo permitir registrar un número de inicio y el mismo número para un mismo tipo de documento; esta no conformidad hace necesario modificar el requisito adicionar talonario según la moneda, tipo de documento y numeración dentro de un mismo ejercicio.

Gestionar caja

En esta funcionalidad se manifestó la inconformidad de que no tenía los requisitos modificar y eliminar una apertura de fondo ya registrada, por lo que se adicionaron los requisitos modificar apertura de un fondo y eliminar apertura, modificándose los requisitos realizar apertura de un fondo y listar los fondos asociados a una caja aperturada.

Gestionar firmas autorizadas

Se adicionó la funcionalidad Gestionar firmas autorizadas, requiriéndose los requisitos Adicionar, Modificar, Eliminar y Listar firmas autorizadas. Además la integración de esta con las funcionalidades Liquidar anticipo de dieta para viaje nacional, Liquidar anticipo de dieta para viaje al exterior, Gestionar anticipo de dieta para viaje nacional y Gestionar anticipo de dieta para viaje al exterior.

Liquidar anticipo de dieta para viaje al exterior

Al liquidar un anticipo de dieta de gastos de viajes, se debe separar la funcionalidad liquidar del contabilizar y adicionar dos nuevos requisitos: modificar y confirmar anticipo de dieta para viaje al exterior.

Liquidar anticipo de dieta para viaje nacional

En esta funcionalidad una de las no conformidades planteadas es al liquidar un anticipo de dieta de gastos de viajes, donde se debe separar la funcionalidad liquidar del contabilizar y adicionar dos nuevos requisitos: modificar y confirmar anticipo de dieta para viaje nacional.

Realizar reembolso y depósito

El sistema no permitía la modificación de un reembolso y depósito ya registrados, por lo que se hizo necesario transformar el requisito modificar reembolso y depósito dentro de las funcionalidades Realizar reembolso y depósito respectivamente; aquí también se señaló la necesidad de visualizar comprobante al contabilizar un depósito y visualizar comprobante al cancelar un depósito en estado "Contabilizado" de la funcionalidad Gestionar depósito; y en la funcionalidad Gestionar reembolso se detectó la ausencia de la visualización del comprobante al contabilizar un reembolso y al cancelar un reembolso en estado "Contabilizado".

Banco: El módulo Banco posibilita el análisis del flujo del capital contable de la entidad. Su propósito principal es gestionar las cuentas bancarias y proporcionar un correcto seguimiento de los talonarios de instrumentos bancarios asociados a cada cuenta bancaria.

El módulo Banco posee un total de 7 funcionalidades, agrupadas en los componentes Configuración, Instrumento, Estado de cuenta, Cierre y Recuperaciones. Las funcionalidades a modificar, partiendo de las no conformidades y pedidos de cambios se concentran en los componentes Instrumento y Estado de cuenta. A continuación se presenta una breve descripción.

Gestionar instrumento

En esta funcionalidad una de las no conformidades detectadas consiste en que la aplicación no permitía la cancelación directa de un instrumento bancario sin necesidad de registrar los datos correspondientes, lo que hizo necesario modificar el requisito Cancelar instrumento. Además se detectó que las operaciones que generaban comprobante de operaciones no lo visualizaban, por tanto se modificó esta funcionalidad al contabilizar un instrumento bancario y al cancelar un instrumento bancario en estado "Contabilizado", visualizándose los comprobantes.

Gestionar estado de cuenta

En esta funcionalidad se señaló que no permitía desde la interfaz de Estado de cuentas realizar las operaciones de Confirmación y Contabilización. Esto provoca la modificación del requisito Adicionar Estado de cuenta y sumar los requisitos de Confirmar y Contabilizar.

Cobros y Pagos: Permite el reconocimiento de las obligaciones de pago y los derechos de cobro contraídos entre la entidad, los clientes y los proveedores; así como al seguimiento de las operaciones asociadas a los derechos y obligaciones y sus estados.(Bailly, 2010)

El módulo Cobro y Pago tiene un total de 23 funcionalidades, agrupadas en 10.componentes. Los cambios en este módulo se concentran en Liquidación, Conciliación, Derecho u obligación, Cambio de cuenta y Conciliación. A continuación se presenta una descripción de las funcionalidades modificadas.

Liquidación de pagos anticipados

Se modifica el requisito Liquidación de derecho u obligación debido a que se debe permitir el flujo completo de una liquidación parcial de un pago anticipado en 2 variantes.

Variante uno: El pago anticipado es mayor que la obligación contraída con el proveedor, se debe generar una devolución de la diferencia del monto. 110/146. Pudiéndose mantener el saldo que queda para otras operaciones con el mismo proveedor.

Variante dos: El pago anticipado es menor que la obligación contraída con el proveedor, se debe generar un pago de la diferencia del monto. 405/110.

Derecho de cobro y Derecho fiscal

Los especialistas funcionales de las entidades consultoras señalaron que se debía permitir la cancelación por expediente de las funcionalidades derecho de cobro y derecho fiscal. En ambos casos esta cancelación debía nutrir al submayor.

Liquidación de cobro anticipado

El sistema debe mostrar el listado de los cobros anticipado pendientes, añadiéndole este requisito a esta funcionalidad del componente Liquidación.

Liquidación de pago anticipado

Se señaló que el sistema debía mostrar el listado de los pagos anticipado pendientes, añadiéndole este requisito a esta funcionalidad del componente Liquidación.

Cambio de cuenta

Se hizo necesario agregarle esta funcionalidad al sistema debido a se debía permitir que los documentos cambiaran de cuenta según la operación que se estuviese realizando.

Conciliación de derecho u obligación

Esta funcionalidad debía permitir buscar derecho de cobro, obligación de pago, pago anticipado, cobro anticipado, derecho fiscal, así como obligación fiscal; además de adicionar conciliación de derecho de cobro, obligación de pago, pago anticipado, cobro anticipado, derecho fiscal.

1.4 Modelo de desarrollo

Para el desarrollo de la solución se empleó el modelo de desarrollo establecido por el CEIGE que propone una estandarización que incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del centro teniendo en cuenta los procesos de *CMMI* (*Capability Maturity Model Integration, del inglés*) nivel 2 para la UCI, detallándose cada uno de los artefactos a generar en cada momento.

1.4.1 Fases del ciclo de vida

Inicio o Estudio preliminar: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto a un alto nivel, la evaluación de la factibilidad del proyecto y el registro de este. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo, y decidir si se ejecuta o no el proyecto. Los objetivos de la fase son:

- Asegurar la factibilidad del proyecto.
- Establecer un plan para la ejecución del proyecto.

Hitos:

- Plan de desarrollo de software.

- Acta de inicio del proyecto firmada.

Desarrollo: 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 refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. El objetivo de esta fase es:

- Obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales.

Hito:

- Producto liberado por entidad certificadora de calidad.

En esta fase se ejecutan las disciplinas Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación. Este modelo de desarrollo en la fase de Diseño propone generar los Diagramas de Clases del Diseño, los Diagramas de Componentes, los Diagramas de Secuencias y el Diagrama de Despliegue. (CEIGE, 2012)

1.5 Tecnologías y herramientas de desarrollo

La selección correcta de las herramientas y tecnologías que se utilizan en el desarrollo de un software se traduce en ahorro de tiempo y trabajo dentro de cualquier proyecto. En la presente investigación se utilizan las herramientas y tecnologías establecidas por el CEIGE para el desarrollo de sus aplicaciones. A continuación se brinda una descripción de las mismas.

1.5.1 Tecnología Ajax

AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. (Garrett, 2005). AJAX es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dados que está basado en estándares abiertos como *JavaScript y Document Object Model* (DOM). Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el

servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor. AJAX es una combinación de tecnologías ya existentes, como JavaScript, DOM, XMLHttpRequest, XHTML, CSS, XML, JSON.

1.5.2 Servidor de aplicaciones web Apache 2.0

Un servidor de aplicaciones es un software que ayuda al servidor Web a procesar las páginas que contienen scripts o etiquetas del lado del servidor.

Apache 2.0

Apache es un servidor Web de tecnología gratuita y de código abierto, permitiendo realizar modificaciones en el código fuente. Es un servidor que corre en una multitud de Sistemas Operativos, que lo hace prácticamente universal. Tiene una alta configuración en la creación y gestión de logs y permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Apache es un servidor altamente configurable de diseño modular que trabaja con gran cantidad de lenguajes de script como Perl, PHP y otros, teniendo todo el soporte que se necesita para tener páginas dinámicas. (Ciberaula, 2010)

1.5.3 Sistemas de Gestores de Base de Datos

Los servidores de bases de datos surgen por la necesidad de manejar grandes y complejos volúmenes de datos, al tiempo que requieren compartir la información con un conjunto de clientes de una manera segura.

PostgreSQL 8.3: Es un Sistema de Gestión de Base de Dato relacional orientado a objetos y libre, publicado bajo la licencia BSD. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales (Martínez, 2010). Dicha comunidad es denominada el PGDG (*PostgreSQL Global Development Group*). Entre sus características se tiene: (González, 2013)

- *Altamente Extensible:* PostgreSQL soporta operadores, funciones, métodos de acceso y tipos de datos definidos por el usuario.

- *Cliente/Servidor:* PostgreSQL usa una arquitectura proceso por usuario cliente/servidor. Esta es similar al método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.
- *API Flexible:* La flexibilidad del API de *PostgreSQL* ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas interfaces incluyen *Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.*

Algunas de sus ventajas son:

- Seguridad en términos Generales.
- Integridad en BD: Restricciones en el Dominio.
- Disparadores (*Triggers*).

1.5.4 Navegador Web

Mozilla Firefox 3.6: Mozilla Firefox es un navegador web libre desarrollado por la Corporación Mozilla. Considerado uno de los navegadores más usado en la actualidad, es multiplataforma y disponible en varias versiones de Microsoft Windows, Mac OS X y GNU/Linux. Su código fuente es software libre, publicado bajo una triple licencia GPL/LGPL/MPL. Mozilla Firefox es compatible con varios estándares web, incluyendo HTML, XML, XHTML, CSS, JavaScript, DOM e imágenes. (López, 2004)

Características del Mozilla Firefox

Sus características incluyen navegación por pestañas, 22 corrector ortográfico, búsqueda progresiva, marcadores dinámicos, un administrador de descargas, navegación privada e integración del motor de búsqueda que desee el usuario. Además se pueden añadir funciones a través de complementos desarrollados por terceros. Proporciona un entorno para los desarrolladores web en el que se puede utilizar herramientas incorporadas, como la Consola de errores, el Inspector DOM, o extensiones como *Firebug*. Es compatible con varios lenguajes web, incluyendo HTML, XML, XHTML. Implementa el sistema SSL/TLS para proteger la comunicación con los servidores web, utilizando fuerte criptografía cuando se utiliza el protocolo https.

1.5.5 Herramienta Case

La Ingeniería de Software Asistida por Computación, CASE por sus siglas en inglés (Computer Aided Software Engineering), define la aplicación de técnicas y métodos por medios de programas, métodos y documentación para comprender y explotar las capacidades de los ordenadores. Las herramientas CASE son las que se encargan de apoyar o automatizar todo el proceso del ciclo de vida de un proyecto de software, siendo imprescindible su uso para la organización y manejo de la información del mismo. Permiten a los administradores del proyecto llevarlo a cabo de forma eficaz y eficiente.

Visual Paradigm 6.4: Es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos que le confiere la característica de ser multiplataforma. Utiliza UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad de forma rápida y barata. Presenta un ambiente gráfico agradable para el usuario. (Visual Paradigm, 2013)

1.5.6 Entorno de desarrollo integrado (IDE)

Un Entorno Integrado de Desarrollo (IDE, Integrated Development Environment) es un sistema que facilita el trabajo del desarrollador de software, integrando sólidamente la edición orientada al lenguaje, la compilación o interpretación, la depuración, las medidas de rendimiento, la incorporación de los fuentes a un sistema de control de fuentes, etc, normalmente de forma modular. (Barahona, y otros, 2007)

Netbeans 6.9: El IDE NetBeans es un entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicaciones que permite el desarrollo rápido en la web, empresa, escritorio y aplicaciones móviles utilizando la plataforma Java, así como *JavaFX, PHP, JavaScript y Ajax, Ruby y Ruby on Rail, Groovy y Grails, y C / C + +.* (NetBeans)

1.5.7 Herramientas de desarrollo colaborativo

Las herramientas de desarrollo colaborativo son un conjunto de componentes que se encargan de llevar un control de versiones de código y presentan elementos básicos como el correo y herramientas de mensajería instantánea que en su conjunto atienden a las necesidades de los integrantes como grupos de desarrollo que pueden o no estar ubicados físicamente en la misma área, es decir, grupos de trabajo compuestos por integrantes trabajando a distancia o en forma remota.

El control de versiones tiene la capacidad de recordar todos los cambios que se hacen tanto en la estructura de directorios como en el contenido de los ficheros. Cuando más de una persona trabaja con los mismos archivos, e inclusive cuando es una sola persona, mantiene cierto control sobre los cambios que se realizan determinando el quién, cuándo y qué. Permite además, tomar decisiones sobre la forma final de un archivo cuando los cambios son realizados por dos personas y facilita recuperar versiones anteriores cuando es necesario por deficiencias detectadas en la última versión. (Serradilla, 2004)

Subversion TortoiseSVN 1.4.5: Es un cliente gratuito de código abierto para el sistema de control de versiones. Administra archivos y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de archivos ordinario, con la diferencia de que recuerda todos los cambios que se hayan hecho a sus archivos y directorios. Esto le permite al desarrollador recuperar versiones antiguas de sus archivos y examinar la historia de cómo y cuándo cambiaron sus datos, y quién hizo el cambio. (Küng, y otros, 2009)

1.6 Lenguaje de modelado.

UML (*Unified Modeling Language*) 2.1: El Lenguaje Unificado de Modelado prescribe no solo la estructura de la aplicación, el comportamiento y la arquitectura, sino también procesos de negocio y estructura de datos. UML, también proporciona una base para el modelo que unifica las etapas de desarrollo e integración de modelos de negocio, a través de modelos arquitectónicos y la aplicación, para el desarrollo, implementación, mantenimiento y la evolución. (Vélez, 2011) El Lenguaje Unificado de Modelado es un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, presenta una vista esencial de la estructura del proyecto, con lo que se visualiza, especifica, construye y documenta el software.

1.7 Lenguajes de programación

Un Lenguaje de Programación es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador poder expresar el procesamiento de datos y sus estructuras en la computadora. Cada lenguaje posee sus propias sintaxis. También se puede decir que un programa es un conjunto de órdenes o instrucciones que resuelven un problema específico basado en un Lenguaje de Programación. (Ramírez, 2011)

1.7.1 Lenguaje de programación en el servidor

Los lenguajes de programación en el servidor se interpretan y ejecutan en el propio servidor web y son los encargados de recibir las peticiones generadas por los usuarios a través de las páginas clientes, y enviarles una respuesta a su petición. Realizan operaciones de acceso a bases de datos, acceso a redes, lógica de negocio, entre otras.

PHP 5.2.6: (Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para desarrollo web y que puede ser incrustado en páginas HTML. El código PHP es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá los resultados de ejecutar el script, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido. Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales. (PHP, 2013)

1.7.2 Lenguaje de programación en el cliente

Los lenguajes de programación en el cliente son ejecutados e interpretados en un navegador web. Son independientes de la plataforma y no requieren de un sistema operativo en específico. Se encargan de realizar las peticiones al servidor a través de las páginas clientes.

HTML: El Lenguaje de Marcado de Hipertexto es el lenguaje que se utiliza para la creación de páginas web. Se compone por una serie de comandos que son interpretados por el navegador. El navegador ejecuta todas las órdenes contenidas en el código HTML, de forma que puede estar capacitado para brindar prestaciones. Este se usa para describir la estructura y el contenido en forma de texto. Se

describe en forma de etiquetas. Puede incluir scripts, los cuales pueden afectar el comportamiento del navegador web. Puede incluir el tratamiento de imágenes y multimedia para acompañar el texto.

1.8 Marco de trabajo

Los *frameworks* son estructuras de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un *framework* se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir las últimas piezas para construir una aplicación concreta. Estos son diseñados para mejorar el desarrollo de software. Son el conjunto de procesos y tecnologías en el cual varios objetos son integrados para dar solución a un problema complejo, en él se estandarizan conceptos, prácticas y criterios para resolver un tipo de problemática en particular. Los *frameworks* permiten acelerar el proceso, reutilizar código ya existente y promover las buenas prácticas en el uso de patrones. (Gutiérrez)

Sauxe 2.0: Sauxe es un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine.

Doctrine: Es un potente y completo sistema ORM para PHP 5.2.3 o superior, que permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos, y el utilizado en una base de datos relacional. Entre otros elementos se tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir, convertir clases (convenientemente creadas) a tablas de una base de datos. En la capa de presentación Sauxe utiliza ExtJS, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una interfaz más amigable y funcional.

ExtJS es una librería Java Script ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas. Una de las grandes ventajas de su uso es que permite crear aplicaciones complejas utilizando componentes predefinidos así como un manejador de *layouts* similar al que provee *Java Swing*, gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en

cada uno. A la hora de desarrollar la documentación de Sauxe, se realizó un análisis de la documentación de ZendFramework, porque fue el *framework* que se tomó para extender sus componentes.

ZendFramework es un framework de código abierto para desarrollar aplicaciones web y servicios web con PHP5, es una implementación que usa código 100% orientado a objetos. La estructura de los componentes es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de ZendFramework conforman un potente y extensible *framework* de aplicaciones web al combinarse, ofrece un gran rendimiento y una robusta implementación MVC, una abstracción de base de datos fácil de usar, y un componente de formularios que implementa la prestación de formularios HTML, validación y filtrado para que los desarrolladores puedan consolidar todas las operaciones usando de una manera sencilla la interfaz orientada a objetos. (Gonzales, 2009)

1.9 Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Una característica que tiene es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. (Hernández, 2013)

1.9.1 Patrones GRASP

GRASP son patrones generales de software para asignación de responsabilidades, es el acrónimo de "GRASP (object-oriented design General Responsibility Assignment Software Patterns)". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. GRASP destaca entre sus patrones principales: Experto, Creador, Controlador, Bajo acoplamiento y Alta cohesión (Hernández, 2013). A continuación se explica brevemente en qué consisten:

Experto: El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la

implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Creador: El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Controlador: Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema, define además el método de su operación. El patrón controlador sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

Bajo acoplamiento: Asigna responsabilidades de modo que se mantenga el bajo acoplamiento. Debe haber pocas dependencias entre las clases, si todas las clases dependen de todas.

Alta cohesión: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Posibilita mejorar la claridad y facilidad con que se entiende el diseño, simplifica el mantenimiento y permite una mayor reutilización.

1.9.2 Patrones GOF

Los patrones de diseño GOF (*Gang of Four*) se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento. (Larman, 2002)

Patrones creacionales

- *Abstract Factory* (fábrica abstracta): permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.

- *Singleton* (instancia única): garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

- *Decorator* (Decorador): Añade funcionalidad a una clase dinámicamente. Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

Patrones estructurales

- *Facade* (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

Patrones de comportamiento

- *Observer (Observador):* Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos. El patrón puede ser usado cuando: Una abstracción tenga dos aspectos, uno dependiente del otro (Se encapsulan estos aspectos por separado). Un cambio en un objeto, requiere cambiar otros objetos. Un objeto debe ser capaz de notificar a otros objetos sin asumir cuales son estos objetos.

- *State:* Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.

- *Memento:* Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.

- *Strategy:* Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

- *Iterator:* Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

- *Template Method:* Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

1.10 Patrón arquitectónico

Un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problema que aparece con frecuencia. (Buschmann, y otros, 1996) Consta de tres partes:

- Contexto: situación de diseño en la que aparece un problema de diseño.
- Problema: conjunto de fuerzas que aparecen repetidamente en el contexto.
- Solución: configuración que equilibra estas fuerzas.

Partiendo de esta definición se puede establecer que los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación. (Buschmann, y otros, 1996)

Patrón Modelo Vista controlador (MVC)

Divide una aplicación interactiva en tres componentes. El modelo contiene la información central y los datos. Las vistas despliegan información al usuario. Los controladores capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario. (Buschmann, y otros, 1996)

1.11 Métricas para la validación del diseño

IEEE (*Institute of Electrical and Electronics Engineers*, por sus siglas en inglés) define las métricas como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Según Pressman (Pressman, 2005), la calidad del diseño se puede evaluar aplicando métricas básicas para la calidad del diseño orientado a objetos. Los atributos de calidad involucrados son:

Responsabilidad: Responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad de implementación: Grado de complejidad que posee la implementación de un diseño de clases específico.

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. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Cantidad de pruebas: Número o grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases, entre otros) diseñado.

1.11.1 Tamaño Operacional de Clase (TOC)

La métrica TOC está dada por la cantidad de métodos pertenecientes a una clase. Está determinada por los atributos: Responsabilidad, Complejidad de implementación y la Reutilización, existiendo una relación directa con los dos primeros e inversa con el último antes mencionado. A continuación se describen los indicadores que se tienen en cuenta en esta métrica (ver tabla 4).

Tamaño Operacional de Clase (TOC)	
Atributo de calidad	**Modo en que lo afecta**
Responsabilidad	El aumento del TOC provoca un aumento de la responsabilidad asignada a la clase.
Reutilización	Un aumento del TOC provoca una disminución en el grado de reutilización de la clase.
Complejidad de Implementación	El aumento del TOC provoca un aumento de la complejidad de implementación de la clase.

Tabla 1. Atributos de calidad que evalúa TOC.

Para la evaluación de cada atributo se definieron los siguientes criterios y categorías de evaluación (ver tabla 5):

	Categoría	Criterio
Responsabilidad	Baja	< =promedio.
	Media	Entre promedio y 2* promedio
	Alta	> 2* promedio
Complejidad 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 de la métrica TOC.

1.11.2 Relación entre clases (RC).

La métrica RC está dada por la cantidad de relaciones de uso existentes entre las clases contenidas en el diseño. Está determinada por los atributos: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas, existiendo una relación directa con los tres primeros e inversa con el último antes mencionado. A continuación se detallan cada uno de estos indicadores. (Ver tabla 3)

Relaciones entre Clases (RC)	
Atributo de calidad	**Modo en que lo afecta**
Acoplamiento	El aumento del RC provoca un aumento del acoplamiento de la clase.
Complejidad del mantenimiento	El aumento del RC provoca un aumento de la complejidad del mantenimiento de la clase.
Reutilización	El aumento del RC provoca una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	El aumento del RC provoca un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 3. Atributos de calidad que evalúa RC.

Para la evaluación de dichos atributos de calidad (ver tabla 4), se definieron los siguientes criterios y categorías de evaluación:

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 de la métrica RC.

1.12 Conclusiones del capítulo

En este capítulo se estudiaron los principales ERP utilizados en Cuba lo que posibilitó obtener un mayor conocimiento acerca de los conceptos y procesos empresariales sobre todo en el área financiera. Se describen las herramientas y metodologías que se establecen en CEIGE para el desarrollo de sus aplicaciones así como su modelo de desarrollo, tomando estos elementos como punto de partida para la generación de artefactos en las fases de diseño e implementación. El estudio de los patrones posibilitó sustentar la base para futuras búsquedas de soluciones a problemas comunes en el desarrollo de software.

CAPÍTULO II: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En este capítulo se describe la solución propuesta, para esto se tomó como punto de partida los requisitos funcionales que fueron modificados previamente. Se generan los artefactos correspondientes a la fase de diseño e implementación tales como Diagrama de clases del diseño, Diagrama de componentes, Diagrama de despliegue, Diagramas de Secuencias y el Modelo de Datos.

2.2 Valoración del análisis

Partiendo de los artefactos obtenidos durante el análisis de los procesos financieros se realiza la valoración de los mismos. Los requisitos entregados por los analistas están claros y se entienden correctamente; se comprueba que las necesidades de los clientes se encuentren cubiertas en las especificaciones de los mismos. La revisión realizada crea una puerta de entrada apropiada y un punto de partida para las actividades de diseño e implementación.

2.2.1 Listado de requisitos funcionales modificados por componentes

Caja

Componente Configuración

- Funcionalidad Gestionar talonario
  - **RF** Adicionar nuevo talonario según la moneda, tipo de documento y numeración dentro de un mismo ejercicio.

- Funcionalidad Gestionar caja
  - **RF** Realizar apertura de un fondo
  - **RF** Modificar apertura de un fondo
  - **RF** Eliminar apertura
  - **RF** Listar fondos asociados a una caja apertura
  - **RF** Asignar talonario a fondo

Componente movimientos

- Funcionalidad Liquidar anticipos de dietas de gastos de viaje al exterior
  - **RF** Modificar anticipo de gasto de viaje al exterior liquidado sin confirmar
  - **RF** Confirmar liquidación de anticipo de gasto de viaje al exterior
- Funcionalidad Liquidar anticipos de dietas de gastos de viaje nacional
  - **RF** Modificar anticipo de gasto de viaje nacional liquidado sin confirmar
  - **RF** Confirmar liquidación de anticipo de gasto de viaje nacional
- Funcionalidad Realizar reembolso
  - **RF** Modificar reembolso
- Funcionalidad Realizar depósito
  - **RF** Modificar depósito
- Funcionalidad Gestionar firma autorizada
  - **RF** Adicionar firma autorizada
  - **RF** Modificar firma autorizada
  - **RF** Eliminar firma autorizada
  - **RF** Listar firma autorizada
  - **RF** Integrar firma autorizada

Banco

Componente estado de cuenta

- Funcionalidad estado de cuenta.
  - **RF** Procesar estado de cuenta.
  - **RF** Contabilizar estado de cuenta.
  - **RF** Adicionar estado de cuenta.

Componente instrumento

- Funcionalidad Gestionar instrumentos bancarios.

- **RF** Cancelar instrumento bancario.
- **RF** Visualizar comprobante al contabilizar un instrumento.
- **RF** Visualizar comprobante al cancelar un instrumento en estado "Contabilizado".

COPA

Componente Liquidación

- Funcionalidad Liquidación de pagos anticipados.
  - **RF** Liquidar pago anticipado.

Componente Derecho u obligación

- Funcionalidad Derecho de cobro.
  - **RF** Cancelar por expediente derecho de cobro.
- Funcionalidad Derecho fiscal.
  - **RF** Cancelar por expediente derecho fiscal.

Componente Cambio de cuenta

- Funcionalidad Cambio de cuenta.
  - **RF** Cambio de cuenta.
  - **RF** Buscar documento a cambiar de cuenta.

Componente conciliación

- Funcionalidad Conciliación de derecho u obligación.
  - **RF** Buscar derecho de cobro a conciliar.
  - **RF** Adicionar conciliación de derecho de cobro.
  - **RF** Buscar obligación de pago a conciliar.
  - **RF** Adicionar conciliación de obligación de pago.
  - **RF** Buscar pago anticipado a conciliar.

- **RF** Adicionar conciliación de pago anticipado.
- **RF** Buscar cobro anticipado a conciliar.
- **RF** Adicionar conciliación de cobro anticipado.
- **RF** Buscar derecho fiscal a conciliar.
- **RF** Adicionar conciliación de derecho fiscal.
- **RF** Buscar obligación fiscal a conciliar.

2.3 Diagrama de clases del diseño

Un diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Describen la estructura de un sistema, donde se muestran las clases, las interfaces y las relaciones que existen entre ellas. (Visconti, y otros, 2010) Esta descripción contiene los siguientes elementos:

- Clases, atributos, métodos y relaciones entre ellas.
- Interfaces con sus componentes.
- Navegabilidad.
- Dependencias.

Diagrama de clases del diseño Gestionar Talonario

La siguiente figura (Ver figura 1) muestra el diagrama de clases identificado para el desarrollo de la funcionalidad Gestionar talonario. En este diagrama se representan las clases "*gestionartalonario.phtml"*, que contienen componentes generados en la librería JavaScript ExtJs y son responsables de visualizar a través del js "*gestionartalonario.js"*, la información necesaria para gestionar los talonarios. Estas clases a la vez incluyen la controladora GesttalonarioControler.php con igual nombre que contiene la implementación de las funcionalidades, y hereda de ZendExt_Controller_Secure para gestionar la seguridad. Los diagramas de clases del diseño para el resto de las funcionalidades se muestran en el Anexo 1.

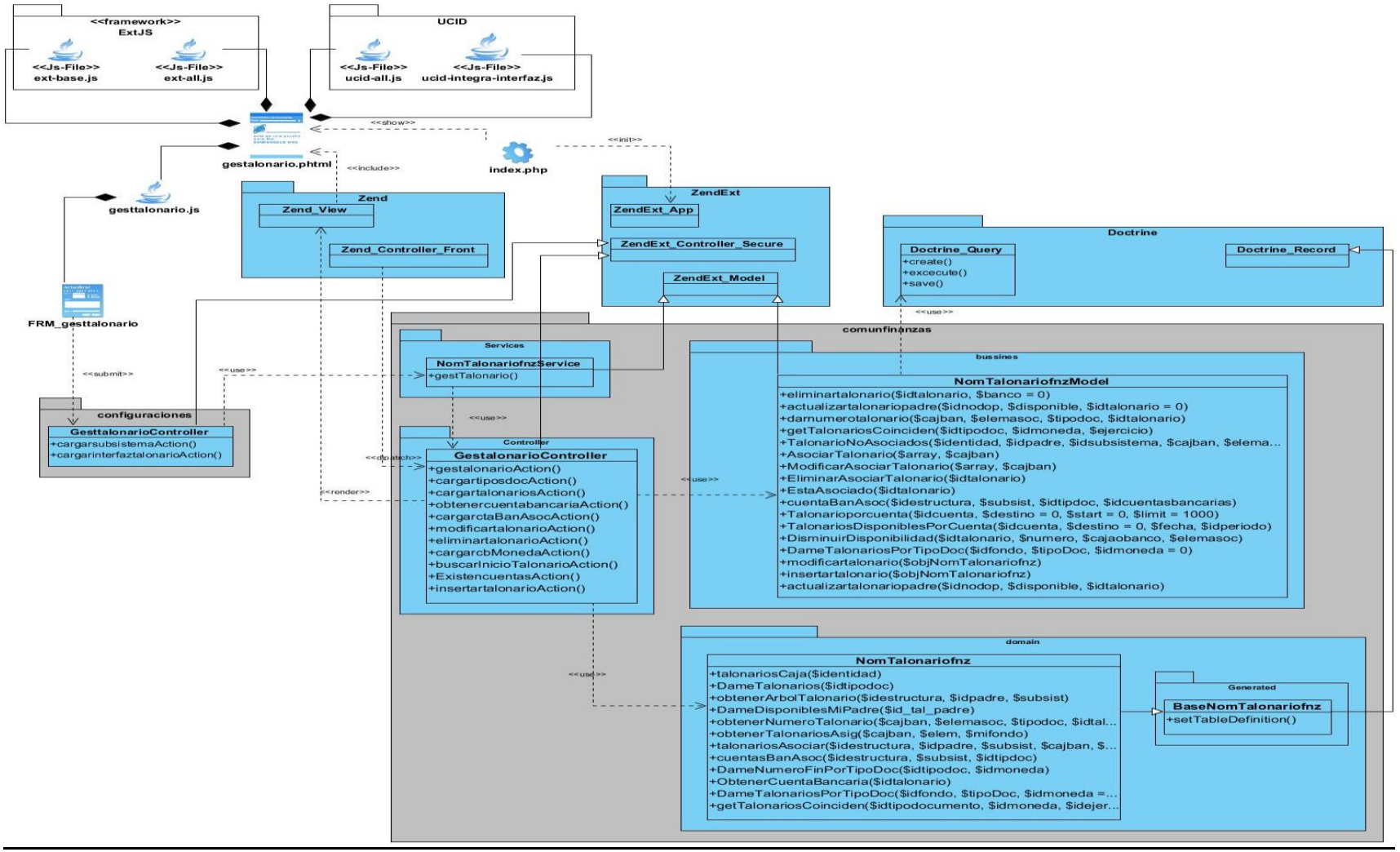

Figura 1. Diagrama de clases del diseño Gestionar talonario.

2.4 Diagrama de componente

El diagrama de componentes del módulo Banco, está conformado por 5 componentes (ver figura 1). El componente Configuración incluye 3 subcomponentes que definen a través de interfaces un conjunto de funcionalidades o servicios de negocio para que puedan ser accedidos por los demás componentes que lo necesitan para su beneficio. El subcomponente Cuenta Bancaria gestiona las cuentas bancarias, configurando el tipo de cuenta que atiende la entidad. Se integra con los subsistemas Caja, Configuración, Estructura y Composición y Contabilidad. El subcomponente Talonario se encarga de la gestión de los talonarios asignados a cada cuenta bancaria.

La gestión de los estados de las cuentas de la entidad y el proceso de conciliación bancaria son realizados por el componente Estado de Cuenta. Las interfaces *EstadoCuentaService* y *ConciliacionService* pertenecientes a este componente brindan servicios que son consumidos por Cuenta Bancaria, Cierre, Recuperaciones y Carga Inicial. Consume servicios del subcomponente Cuenta Bancaria del propio módulo y del componente de integración *comunfinanzas*.

El componente Cierre realiza el cierre diario de período contable y de ejercicio fiscal. No ofrece servicios, pero consume de los componentes Cuenta Bancaria, Instrumento, Estado de Cuenta y del componente de integración *comunfinanzas*. El componente Recuperaciones gestiona los reportes del módulo. No ofrece servicios, pero consume de los componentes Cuenta Bancaria, Estado de Cuenta y del componente de integración *comunfinanzas*. Los diagramas de componentes para el resto de los módulos se muestran en el Anexo 2.

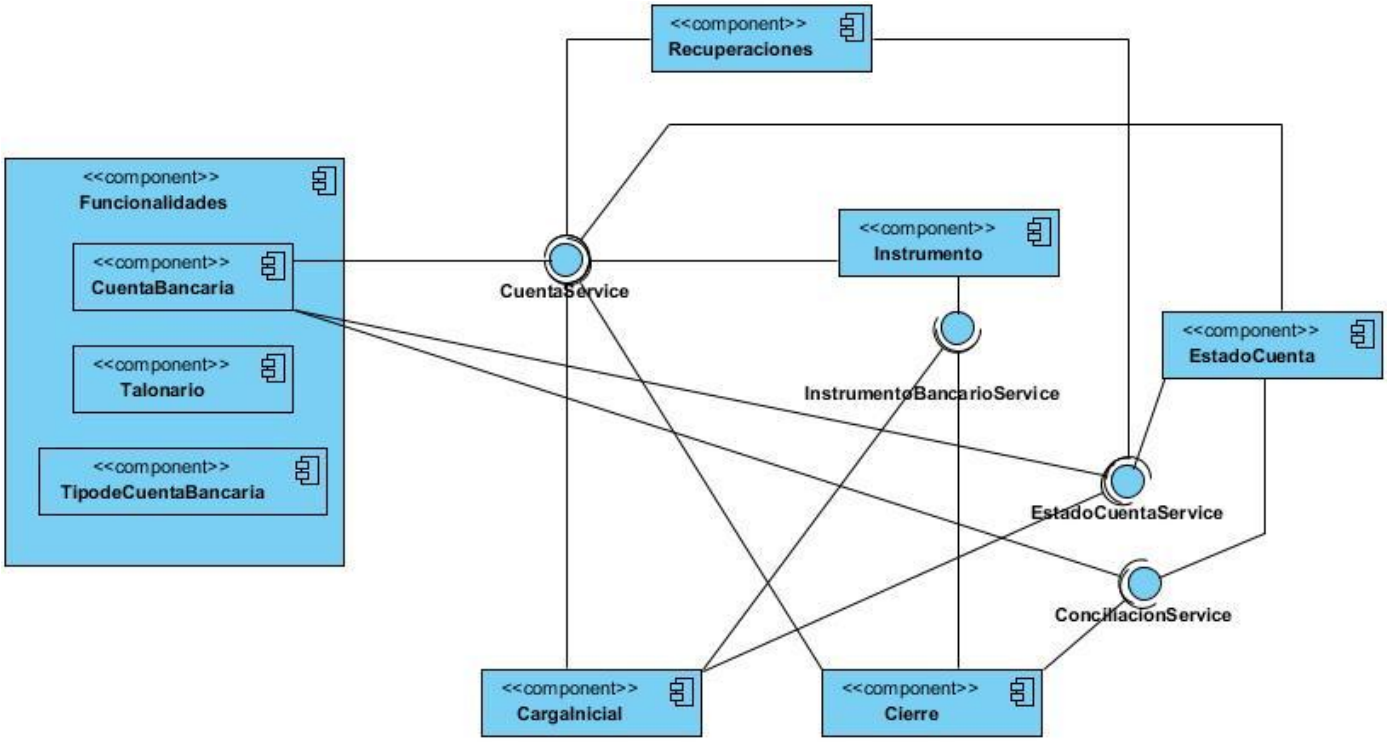

Figura 2. Diagrama de componentes. Módulo Banco.

2.5 Diagrama de secuencia

Otros de los artefactos generados durante el diseño del sistema fueron los diagramas de secuencia correspondientes a cada uno de los requisitos funcionales modificados, estos constituyen una forma efectiva para modelar la interacción entre los diferentes objetos del sistema, mostrando gráficamente las interacciones del actor y de las operaciones a que dan origen. A continuación se muestra el diagrama de secuencia (Ver Figura 3) correspondiente al requisito Adicionar talonario, el resto de los diagramas remitirse al Expediente de Proyecto.

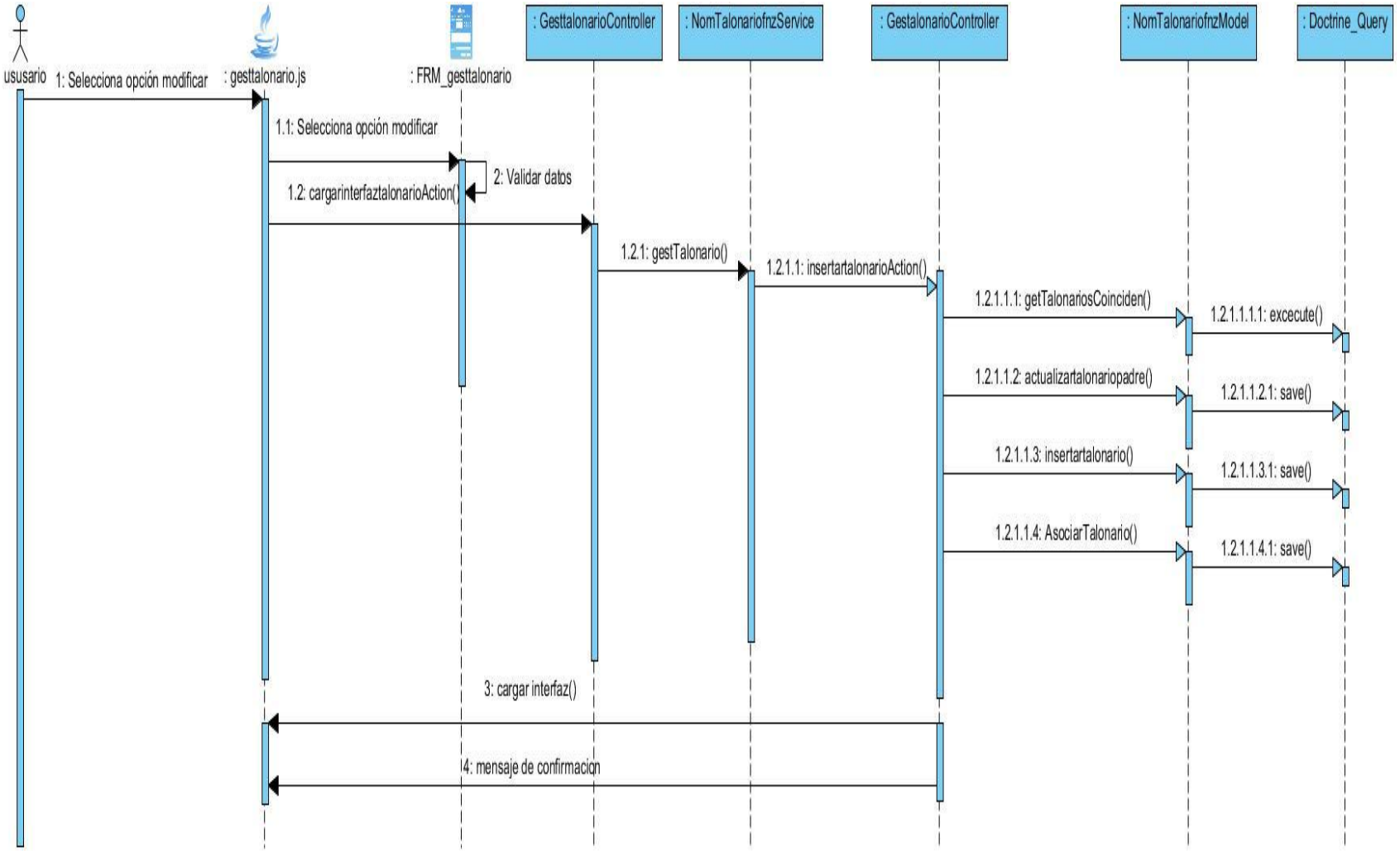

Figura 3. Diagrama de Secuencia Adicionar Talonario

2.6 Modelo de datos

La base de datos está organizada en esquemas que siguen el siguiente patrón: un esquema que contiene las configuraciones de todo el sistema, un esquema encargado de contener información común a todos los subsistemas (por ejemplo: los nomencladores) y el esquema que gestiona el negocio en sí de cada subsistema, pudiendo ser uno o varios según la complejidad del negocio.

Esquema Caja: Es el encargado de persistir la configuración inicial de una caja de un sistema contable, así como el conjunto de operaciones que se realizan sobre los distintos fondos monetarios con que opera esta, estas operaciones pueden ser tanto el arqueo de dichos fondos como los distintos movimientos que se realizan. El siguiente modelo de datos representa las principales tablas del esquema caja, para una mayor profundización remitirse al Expediente de Proyecto.

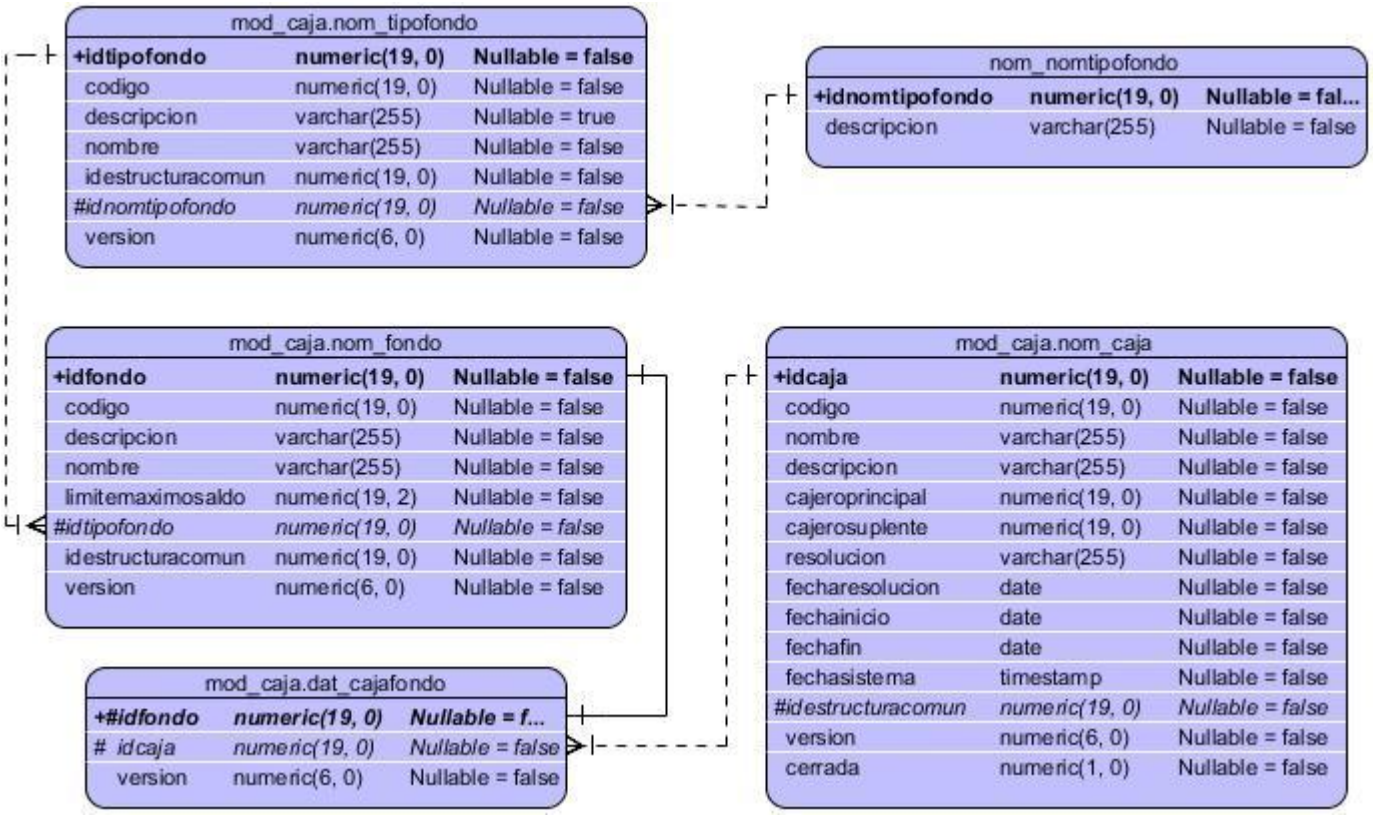

Figura 4. Diagrama del Modelo de datos "Módulo Caja".

2.7 Patrones de diseño presentes en la solución

GRASP

Experto: Está presente en todos los módulos del subsistema, con la asignación de responsabilidades específicas por cada una de las clases del sistema. Los componentes cuentan con clases controladoras, modelos y entidades, con funciones específicas atendiendo a la actividad que realiza y los procesos que gestiona. Además, se modeló una clase entidad por cada tabla de la base de datos, posibilitando el trabajo específico y directo con el experto en la información.

Creador: Es útil contar con el principio general para la asignación de responsabilidades de creación porque permite que el diseño pueda soportar bajo acoplamiento, mayor claridad, encapsulación y reutilización. En el subsistema se evidencia este patrón en cada módulo, las clases controladoras son responsables de crear el objeto de las modelos, y estas a su vez de las entidades.

Controlador: Se utilizan cuando la aplicación es muy extensa, de esta forma, en vez de tener un solo controlador y saturarlo, se tienen clases "*Controllers"*, que son controladores más pequeños especializados en las funcionalidades de cada componente. Por ejemplo en Gestionar talonario se tienen dos clases controladoras, una en el componente Configuración de Caja y otra en el componente de integración *comunfinanzas.* Se trata de dividir las responsabilidades especializando cada clase controladora.

Bajo acoplamiento: Los componentes en el subsistema fueron diseñados bajo este principio, porque solo establecen las relaciones necesarias entre ellos. La base de datos fue definida de modo que entre las tablas existiera dependencia mínima, haciéndolas más independientes y reutilizables para reducir el impacto de los cambios y acrecentar la oportunidad de una mayor productividad.

Alta cohesión: En el subsistema existe afinidad entre cada clase y los métodos que implementan, estas poseen responsabilidades vinculadas acordes a la información que controlan; y colaboran con otros objetos para compartir el esfuerzo si la tarea es grande, facilitando su mantenimiento y reutilización.

GoF

Proxy: Este patrón se evidencia mediante el uso de Doctrine para el trabajo con la base de datos. El uso de este patrón garantizó asignar a cada clase la responsabilidad que le corresponde, obtener el menor número de relaciones y dependencias entre clases y aumentar las posibilidades de reusabilidad de las mismas.

2.8 Conclusiones del capítulo

Con el desarrollo de este capítulo se realizó el diseño de la propuesta de solución generando los artefactos Diagrama de Clases del Diseño, Diagrama de Componentes, Diagrama de Secuencia, Modelo de Datos, los cuales constituyen una puerta de entrada apropiada para la implementación de las mejoras funcionales del subsistema Finanzas del Sistema Integral de Gestión Cedrux.

CAPÍTULO III VALIDACIÓN DE LA SOLUCIÓN

3.1 Introducción

La complejidad en la construcción de sistemas en la industria de la informática aumenta el riesgo de que permanezcan errores aún después de terminado. Afirmar que un software se encuentra plenamente libre de errores es casi imposible, pero existen formas y métodos de acercarse lo más posible a la solución óptima. En el presente capítulo se realiza la validación del diseño aplicando las métricas (TOC) y (RC) Además se realizan pruebas de caja negra con el objetivo de verificar la funcionalidad y estructura de cada componente desarrollado.

3.2 Métrica para la evaluación del diseño Tamaño Operacional de Clase (TOC)

El siguiente gráfico muestra el porciento de clases en los diferentes intervalos definidos por la métrica.

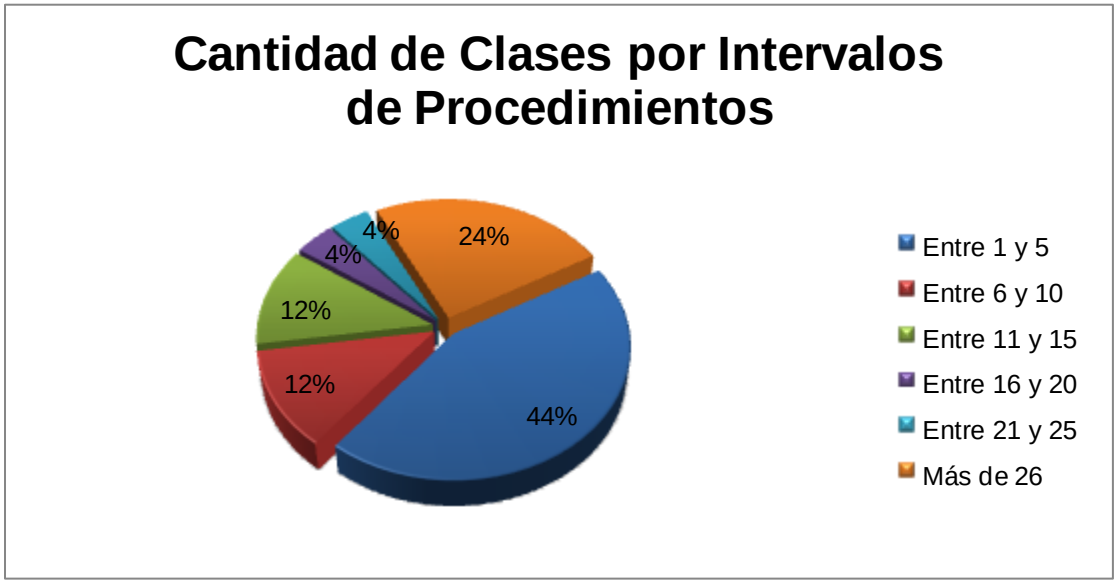

Figura 5. Representación en % de los resultados obtenidos, agrupados en los intervalos definidos según la métrica TOC.

En la Figura 6 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. Este gráfico muestra un resultado satisfactorio pues el 62% de

las clases poseen una baja responsabilidad. Esta característica permite que en caso de fallos, como la responsabilidad está distribuida de forma equilibrada ninguna clase es lo suficientemente crítica como para dejar al sistema fuera de servicio.

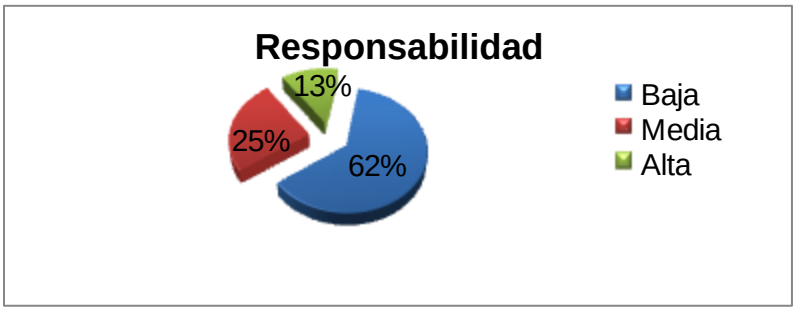

Figura 6. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

En la Figura 7 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación. Este gráfico muestra un resultado satisfactorio pues más del 50% de las clases poseen una baja complejidad de implementación. Esta característica permite mejorar el mantenimiento y soporte de estas clases.

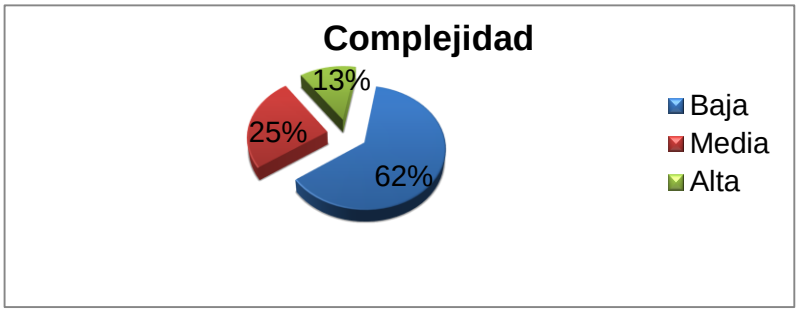

Figura 7. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

La Figura 8 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización. El diseño del componente tiene un grado de eficiencia aceptable pues solamente el 12% del total de las clases poseen una baja reutilización.

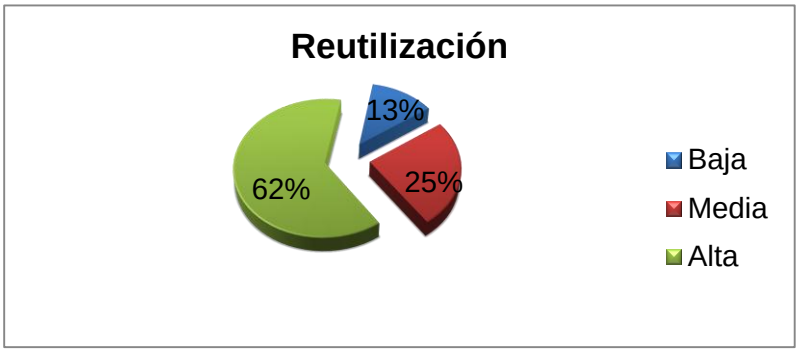

Figura 8. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Analizando los resultados obtenidos de la métrica TOC, se puede concluir que el diseño del componente tiene una calidad aceptable teniendo en cuenta los resultados arrojados por los atributos analizados. Solo el 13% de las clases presentan una alta responsabilidad, una alta complejidad de implementación y una baja reutilización, lo cual demuestra que el resultado es satisfactorio.

3.3 Métrica para la evaluación del diseño Relaciones entre Clases (RC)

El gráfico de la Figura 9 refleja que el 57% de las clases tienen cero dependencias, y el 19% una dependencia con otra clase. Este resultado es positivo pues demuestra que el 76% de las clases se encuentran dentro de los niveles aceptables de calidad.

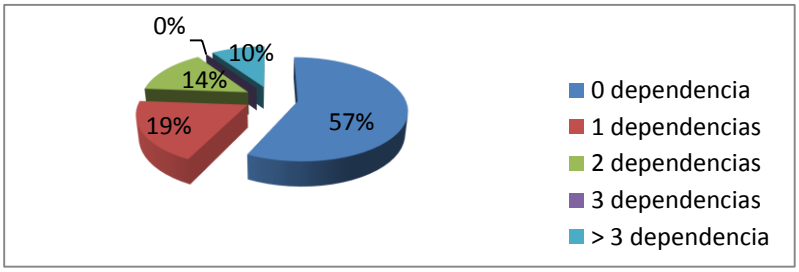

Figura 9. Representación en % de los resultados obtenidos, agrupados en los intervalos definidos según la métrica RC.

La Figura 10 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento. Se evidencia un bajo acoplamiento entre las clases pues el 27% de las clases no presentan dependencias con otra y el 44% una dependencia con otra. Este resultado es muy

favorable para el diseño del componente pues al existir poca dependencia entre las clases aumenta el grado de reutilización del componente.

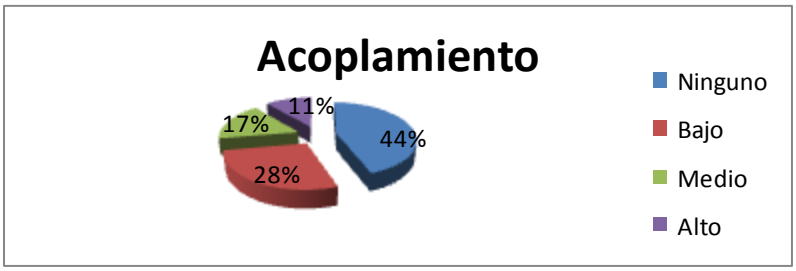

Figura 10. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Acoplamiento.

La Figura 11 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. El gráfico refleja un resultado aceptable del atributo pues solamente el 11% de las clases presentan una alta complejidad de mantenimiento.

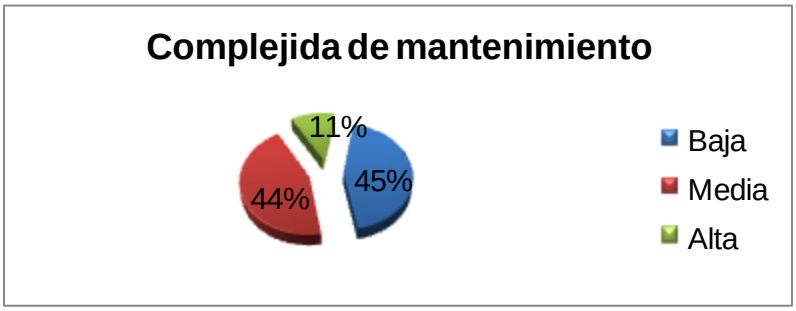

Figura 11. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de mantenimiento.

La Figura 12 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas. Evidenciando que solamente al 11% de las clases hay que hacerle un elevado número de pruebas.

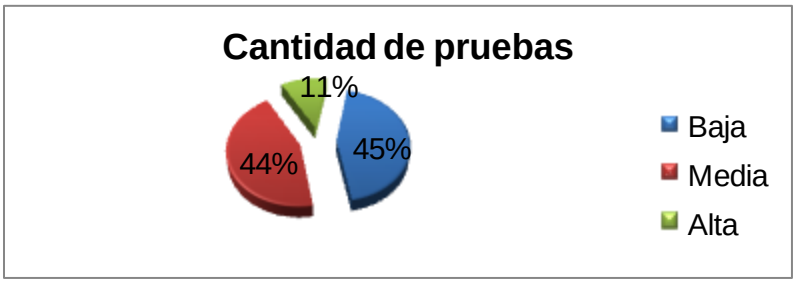

Figura 12. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Cantidad de pruebas.

La Figura 13 muestra la representación de la incidencia de los resultados de la evaluación del atributo reutilización. Esto evidencia que solo el 11% de las clases poseen una alta reutilización lo que es un factor fundamental que debe ser tenido en cuenta en el desarrollo de software.

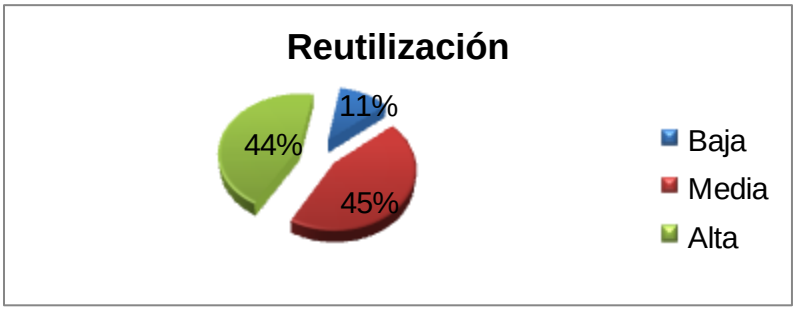

Figura 13: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño de las mejoras funcionales en el subsistema Finanzas en su versión 1.1 tiene una calidad aceptable por todo lo antes expuesto.

3.4 Pruebas de Software

Las pruebas de software son un instrumento para determinar el status de la calidad de un software. Tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos. (pruebasdesoftware.com, 2005)

3.4.1 Estrategia de pruebas

Para realizar el proceso de la aplicación de pruebas se define una estrategia de pruebas regida según el modelo de desarrollo establecido, con el fin de garantizar la calidad del software. Se concretan los casos de pruebas, a partir de la construcción de todos los posibles caminos de ejecución, o escenarios de cada componente desarrollado, basados en los requisitos implementados, comparando cada funcionalidad descrita con lo implementado para verificar hasta qué punto concuerdan y cumplen con las necesidades del cliente. Se obtiene como resultado un listado final con los casos de prueba identificados a partir de los posibles escenarios, los resultados esperados para cada caso y las condiciones o valores requeridos para la ejecución de los distintos escenarios. Se diseñaron un total de 35 casos de prueba. De los cuales hay 9 diseños de casos de pruebas liberados por calidad interna del centro, el resto fueron liberados por el equipo de analistas de la línea que constituyen el cliente.

3.4.2 Pruebas de Caja Negra

Las pruebas de caja negra son las que se aplican a la interfaz del software y se centran en los requisitos funcionales del sistema; permitiendo derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. (Pressman, 2005)

Los casos de prueba están basados en diferentes entradas que puede recibir el sistema con sus correspondientes valores de salida. Se centran en realizar pruebas de software para comprobar su funcionalidad. Los casos de prueba demuestran que:

- Las funciones del software son operativas.
- Las entradas se aceptan de la forma adecuada produciendo el resultado esperado.
- La integridad de la información externa (por ejemplo archivos de datos) se mantiene.

Para verificar que la aplicación cumpla con los requisitos establecidos por el cliente se diseñan los casos de prueba y se realizan las pruebas internas y de aceptación del cliente. A continuación se especifica el caso de prueba “Adicionar talonario” el cual registra los talonarios que corresponden a los documentos primarios que se utilizan al realizar las distintas operaciones sobre los fondos. Para ver el diseño de los demás requisitos funcionales remitirse al Expediente de Proyecto.

Condiciones de ejecución

- Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar esta acción.
- Se debe seleccionar el subsistema de Finanzas.
- Se selecciona la opción **Caja/Configuración/Gestionar talonarios**.
- Se selecciona la opción “Talonario de caja”.
- Se presiona el botón **Adicionar**.

Requisito a probar.

Nombre del requisito	Descripción general	Criterio	Flujo del escenario
1: Adicionar talonario.	El sistema permite adicionar talonarios al sistema	EP 1.1: Adicionar talonario introduciendo los datos correctamente.	- Se introducen correctamente todos los datos. - Se presiona el botón Aceptar. - El sistema muestra el mensaje: “Se adicionó el talonario satisfactoriamente”
		EP 1.2: Adicionar talonario introduciendo datos inválidos.	- Se introducen datos inválidos. - Se presiona el botón Aceptar. - Se muestra el mensaje: “Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s)”
		EP 1.3: Adicionar talonario dejando campos vacíos.	- Se introducen los datos dejando campos vacíos que requieren ser llenados. - Se presiona el botón **Aceptar**.

		- Se muestra el mensaje: "Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s)"
	EP 1.4: Adicionar talonario cuando ya existe uno igual.	- Se introducen los datos del talonario que coinciden con uno ya existente. - El sistema verifica que no existe en el mismo ejercicio económico un talonario con el mismo documento, moneda y número. - El sistema muestra un mensaje: "No se puede insertar el talonario, ya existe."
	EP 1.5: Cancelar.	- Se introducen o no los datos. - Se presiona el botón **Cancelar**

Tabla 5. Descripción del caso de prueba para el requisito Adicionar talonario.

Descripción de variable.

No	Nombre del campo	Clasificación	lido	Inválido
1	Tipo documento	Lista desplegable (Combobox).	N/A	N/A
2	Moneda	Lista desplegable (Combobox).	NA	NA
3	Número inicio	Campo de texto	1	(sgh &/% /*)
4	Número fin	Campo de texto	20	(sgh &/% /*)
5	Disponible	Campo de texto	N/A	N/A

Tabla 6. Descripción de las variables del diseño de caso de prueba Adicionar talonario.

Juego de datos a probar.

Id del escenario	Escenario	Tipo documento	Moneda	Número inicio	Número fin	Disponibles	Respuesta del sistema
EP 1.1	Adicionar talonario introduciendo los datos correctamente	V(Recibo de efectivo)	V(CUP)	V(1)	V(10)	V(10)	Una vez presionado el botón **Aceptar** del mensaje de información: "Se adicionó el talonario satisfactoriamente.", este se
EP 1.2	Adicionar talonario introduciendo datos inválidos.	N/A	N/A	I(hfg##hg)	V(10)	N/A	El sistema no permite la inserción de letras y símbolos en este campo y se muestra la siguiente alerta: "Este campo es obligatorio". Se muestra el mensaje: "Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s)

							”
		N/A	N/A	V(1)	I(hfg##hg)	N/A	El sistema no permite escribir en este campo y se muestra la siguiente alerta: “Este campo es obligatorio”. Se muestra el mensaje: “Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s) ”
EP 1.3	Adicionar talonario dejando campos vacíos.	I(Vacío)	NA	NA	NA	NA	El sistema muestra la siguiente alerta: “Este campo es obligatorio.”. Se muestra el mensaje: “Por favor verifique nuevamente que hay campo(s)

					con valor(es) incorrecto(s) ”
V(Recibo de efectivo)	I(Vacío)	NA	NA	NA	El sistema muestra la siguiente alerta: “Este campo es obligatorio.” Se muestra el mensaje: “Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s) ”
V(Recibo de efectivo)	V(CUP)	NA	I(Vacío)	NA	El sistema muestra la siguiente alerta: “Este campo es obligatorio.”. Se muestra el mensaje: “Por favor verifique nuevamente que hay campo(s) con valor(es) incorrecto(s) ”

EP 1.4	Adicionar talonario cuando ya existe uno igual.	V(Recibo de efectivo)	V(CUP)	V(1)	V(4)	V(4)	El sistema muestra el siguiente mensaje: "No se puede insertar el talonario, ya existe".
EP 1.5	Cancelar.	NA		NA	NA	NA	Se cierra la ventana.

Tabla 7. Juego de datos caso de prueba Gestionar talonario.

3.5 Resultados de las pruebas internas

Para comprobar la calidad y funcionalidad del software desarrollado y con el empleo de los diseños de casos de pruebas para la aplicación, se realizaron dos iteraciones de revisiones por el grupo de calidad del Centro. Durante la realización de las pruebas se detectaron un conjunto de no conformidades y posteriormente se procedió a su solución.

3.5.1 Primera iteración de pruebas internas de calidad

Durante la primera iteración se detectaron siete no conformidades, de ellas tres fueron del módulo Caja, y dos de los módulos Banco y Cobro y Pago. A continuación se muestra un gráfico con el porciento que representan del total de no conformidades por tipo:

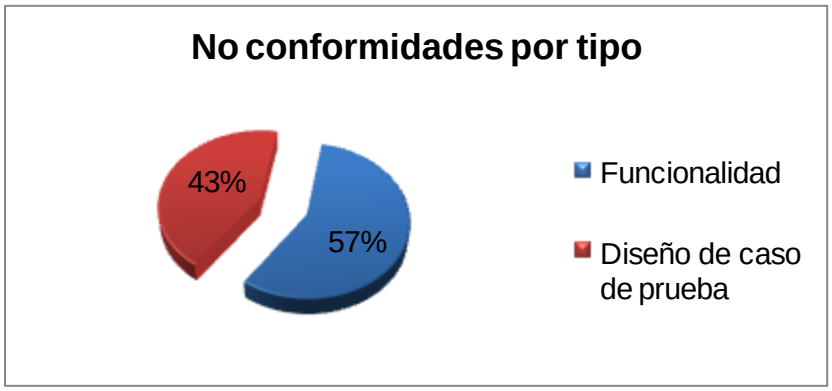

Tabla 8. Porciento que representan la cantidad de no conformidades por tipo respecto al total de la primera iteración.

Las no conformidades referentes a funcionalidad y diseño de caso de prueba representan el 57 y el 43 porciento respectivamente. Se resolvieron en su totalidad para un 100 porciento de solución.

3.5.2 Segunda iteración de pruebas internas de calidad.

En la segunda iteración no se detectaron no conformidades, liberándose el producto con éxito emitiéndose la carta de liberación por el grupo de calidad del CEIGE.

3.6 Resultados pruebas de aceptación del cliente

En una primera etapa todos los casos de prueba no pudieron pasar por el equipo de calidad del centro, por lo que se decidió realizar las pruebas de aceptación del cliente. Donde se comprueba que el software cumple con los requisitos funcionales especificados.

3.6.1 Primera iteración de las pruebas aceptación del cliente

En esta primera iteración fueron detectadas cinco no conformidades, dos de los módulos Caja y Cobro y Pago y una de Banco. A continuación se muestra un gráfico con el porciento que representan del total de no conformidades por tipo:

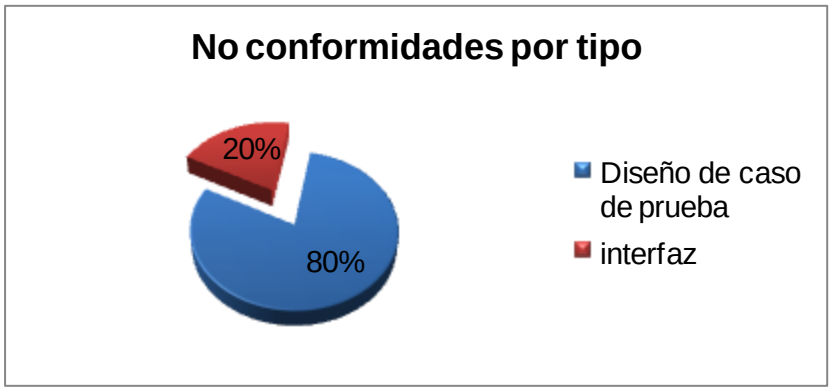

Figura 14: Porciento que representan la cantidad de no conformidades por tipo respecto al total de la primera iteración

Las no conformidades referentes a diseño de caso de prueba representan en mayor medida el nivel de afectación en la aplicación para un 80%, resolviéndose completamente las no conformidades detectadas.

3.6.2 Segunda iteración de las pruebas de aceptación del cliente

En la segunda iteración no se detectaron no conformidades, liberándose el producto con éxito, lográndose de esta manera la satisfacción de las necesidades del cliente.

3.7 Conclusiones

En el presente capítulo se aplicaron las métricas TOC y RC, y a partir de los resultados obtenidos se arribó a la conclusión de que el diseño es aceptable. Las pruebas de Caja negra arrojaron como resultado que las funcionalidades implementadas mostraron un correcto funcionamiento, respondiendo adecuadamente a los requisitos funcionales y garantizando la satisfacción plena de las demandas del cliente.

CONCLUSIONES GENERALES

La fundamentación de la investigación sustentó la propuesta de solución precisando los conceptos fundamentales para la investigación. Las características del funcionamiento de los distintos Sistemas de Planificación de Recursos Empresariales más utilizados en nuestro país ayudaron a dar solución a las mejoras funcionales de subsistema Finanzas en su versión 1.1; también el estudio de las herramientas, lenguajes de programación y tecnologías utilizadas, así como del modelo de desarrollo adoptado para desarrollar la propuesta de solución.

Con el diseño de la propuesta de solución, generando los artefactos que propone el Modelo de Desarrollo, se implementó la solución de las mejoras funcionales del subsistema Finanzas del Sistema Integral de Gestión Cedrux para satisfacer sus requisitos funcionales.Con la ejecución de las métricas de diseño para evaluar indicadores del diseño, se llegó a la conclusión de que el diseño elaborado cumple con los diferentes aspectos medidos a partir de las métricas. Se realizaron también pruebas de caja negra que permitieron evaluar funcionalmente el software. Igualmente, las funcionalidades implementadas mostraron el correcto funcionamiento, respondiendo adecuadamente a los requisitos funcionales y garantizando la satisfacción plena de las demandas del cliente.

GLOSARIO DE TÉRMINOS

Enterprise Resource Planning: Sistema de información integral que permite la ejecución y automatización de los procesos de negocio de todas las áreas funcionales de un modo combinado.

CONAVANA: Consultorías y Avalúos, es una sociedad mercantil de capital 100% cubano, fundada en 1996 cuya misión es prestar servicios profesionales de avalúos, consultorías económicas y de gestión empresarial, al empresariado cubano y extranjero integrando para ello la experiencia y profesionalidad de su equipo de trabajo, con metodologías y técnicas reconocidas y aceptadas nacional e internacionalmente.

Interaudit (Auditores y consultores): Fue creada el 26 de marzo de 1993. Su actividad fundamental es prestar servicios especializados a empresas mixtas, personas naturales y jurídicas extranjeras y ciudadanos cubanos radicados en el exterior, en pesos convertibles y a personas jurídicas cubanas en pesos cubanos.

CANEC: Consultoría Económica CANEC S.A. fue fundada en 1992 por la Asociación Nacional de Economistas y Contadores de Cuba (ANEC), organización no gubernamental con estatus consultivo especial del Consejo Económico y Social de Naciones Unidas (ECOSOC), y que preside la Asociación de Economistas de América Latina y el Caribe (AEALC).

ORM (Object-relational mapping): Técnica de programación que permite convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional. El mapeo objeto-relacional posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

SOA (Service-oriented architecture): Arquitectura desacoplada de componentes de software que proveen funciones específicas (proveedor) y que pueden ser invocadas por otros componentes (consumidor) independientemente de la plataforma en que se encuentren ambos

BIBLIOGRAFÍAS

Arias, Taimé Ramos y Salas, Pedro Antonio Torres. 2010. *Diseño e implementación del módulo Banco del Sistema Integral de Gestión CEDRUX.* La Habana : UCI, 2010.

Bailly, Gregseen Wilson. 2010. *Vistas de arquitectura de sistema y datos de la solución Finanzas del proyecto ERP Cuba.* La Habana : UCI, 2010.

Barahona, Jesús M. González, Robles, Gregorio y Pascual, Joaquín Seoane. 2007. BerliOS. [En línea] 2007. [Citado el: 5 de Abril de 2013.] http://curso-sobre.berlios.de/introsobre/2.0.1/sobre.html/sec-ide.html.

Baryolo, Oiner Góomez. 2008. *Plantilla Registro de la Propiedad intelectual (Sauxe).* 2008.

Buschmann, F., y otros. 1996. *Pattern – Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.

CEIGE. 2012. *MODELO DE DESARROLLO DE SOFTWARE.* La Habana : UCI, 2012.

Ciberaula. 2010. Una Introducción a Apache. [En línea] 2010. [Citado el: 16 de Abril de 2012.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

Garrett, Jesse James. 2005. adaptivepath. [En línea] 18 de Febrero de 2005. [Citado el: 5 de Mayo de 2013.] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.

Gonzales, Benjamín. 2009. *Zend Framework. Manual en Español.* s.l. : Zend Technologies Inc, 2009.

González, Carlos D. 2013. Usabilidad Web. [En línea] Junio de 2013. [Citado el: 10 de Junio de 2013.] http://www.usabilidadweb.com.ar/postgre.php.

González, Liset Feria. 2010. *Vista de arquitectura de seguridad del proyecto ERP-Cuba.* La Habana : UCI, 2010.

Guevara Alvarez, Lianet, y otros. 2012. *SISTEMA INTEGRAL DE GESTIÓN CEDRUX: SUBSISTEMA BANCO.* La Habana : UCI, 2012.

Gutiérrez, Javier J. Lenguajes y Sistemas Infórmaticos. [En línea] [Citado el: 5 de Mayo de 2013.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

Hernández, Ing. Dayana C. Tejera. 2013. Entorno Virtual de Aprendizaje. [En línea] 2013. [Citado el: 9 de Mayo de 2013.] http://eva.uci.cu/course/view.php?id=161.

Hurtado, Jorge Luis Naranjo. 2010. *Vista de la arquitectura de sistema y datos del subsistema Capital Humano del proyecto ERP-Cuba.* La Habana : UCI, 2010.

Jiménez, Anierys Delgado y Arce, Nereyda González . 2010. *Descripción de los procesos de negocio de los subsistemas Finanzas y Logística de CEDRUX.* La Habana : UCI, 2010.

Jiménez, Anierys Delgado y Arce, Nereyda González. 2010. *Descripción de los procesos de negocio de los subsistemas Finanzas y Logística de CEDRUX.* La Habana : UCI, 2010.

Küng, Stefan, Onken, Lübbe y Large, Simon. 2009. TortoiseSVN. [En línea] 20 de Enero de 2009. [Citado el: 30 de Abril de 2013.] http://tortoisesvn.net/docs/release/TortoiseSVN_es/index.html.

Larman, Craig. 2002. *UML y Patrones.* 2002.

López, Alejandro Cadavid. 2004. Maestros del web. [En línea] 16 de Junio de 2004. [Citado el: 10 de Febrero de 2013.] http://www.maestrosdelweb.com/editorial/firefox/.

Manfugás, Lisandra Rowe. 2011. *Propuesta de perfeccionamiento funcional para el módulo Banco del subsistema Finanzas de Cedrux.* La Habana : UCI, 2011.

Martinez, Rafael. 2010. PostgresSQL-es. [En línea] 2 de Actubre de 2010. [Citado el: 10 de Abril de 2013.] http://www.postgresql.org.es/sobre_postgresql.

Mifsud, Elvira. 2008. Observatorio Tecnólogico. [En línea] 21 de Abril de 2008. [Citado el: 20 de Mayo de 2013.] http://recursostic.educacion.es/observatorio/web/es/software/servidores/580-elvira-mifsud.

NetBeans. NetBeans. [En línea] [Citado el: 13 de Abril de 2013.] https://netbeans.org/community/releases/69/.

PHP. 2013. PHP. [En línea] 07 de Junio de 2013. [Citado el: 13 de Junio de 2013.] http://php.net/manual/es/intro-whatis.php.

Polanco, Martha Elena Hevia. 2010. *Vistas de arquitectura de sistema y datos de los Subsistemas Contabilidad y Costos y Procesos del proyecto ERP Cuba.* La Habana : UCI, 2010.

Ramírez, Dr. Vicente Rico. 2011. Instituto Tecnolóogico de Celaya. [En línea] Septiembre de 2011. [Citado el: 16 de Abril de 2013.] http://www.iqcelaya.itc.mx/~vicente/Programacion.html.

Rivas, Jeanne Omara González. 2010. *Vistas de arquitectura de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda del proyecto ERP Cuba.* La Habana : UCI, 2010.

Serradilla, Juan Luis. 2004. ATICA. [En línea] 16 de Junio de 2004. [Citado el: 18 de Abril de 2013.] http://www.um.es/atica/control-de-versiones-con-subversion-3#d0e89.

Valladares, Tania Teresa Loureiro y Baez, Iyugnis Leyva. 2012. *SUBSISTEMAS DE GESTIÓN BANCO Y CAJA DEL SISTEMA INTEGRAL DE GESTIÓN CEDRUX.* La Habana : UCI, 2012.

Vélez, Jorge Carlos Driggs. 2011. *Diseño e Implementación de la nueva versión del Módulo Banco del Sistema Integral de Gestión Cedrux.* La Habana : UCI, 2011.

Visconti, Marcello y Astudillo, Hernán. 2010. Departamento de Informática. Universidad Técnica de Federico de Santa Maria. [En línea] 2010. [Citado el: 20 de Abril de 2013.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/10-DisenoOO.pdf..

Visual Paradigm. 2013. Visual Paradigm. [En línea] 21 de Enero de 2013. [Citado el: 9 de Febrero de 2013.] http://www.visual-paradigm.com/product/vpuml/.

ANEXOS

Anexo 1

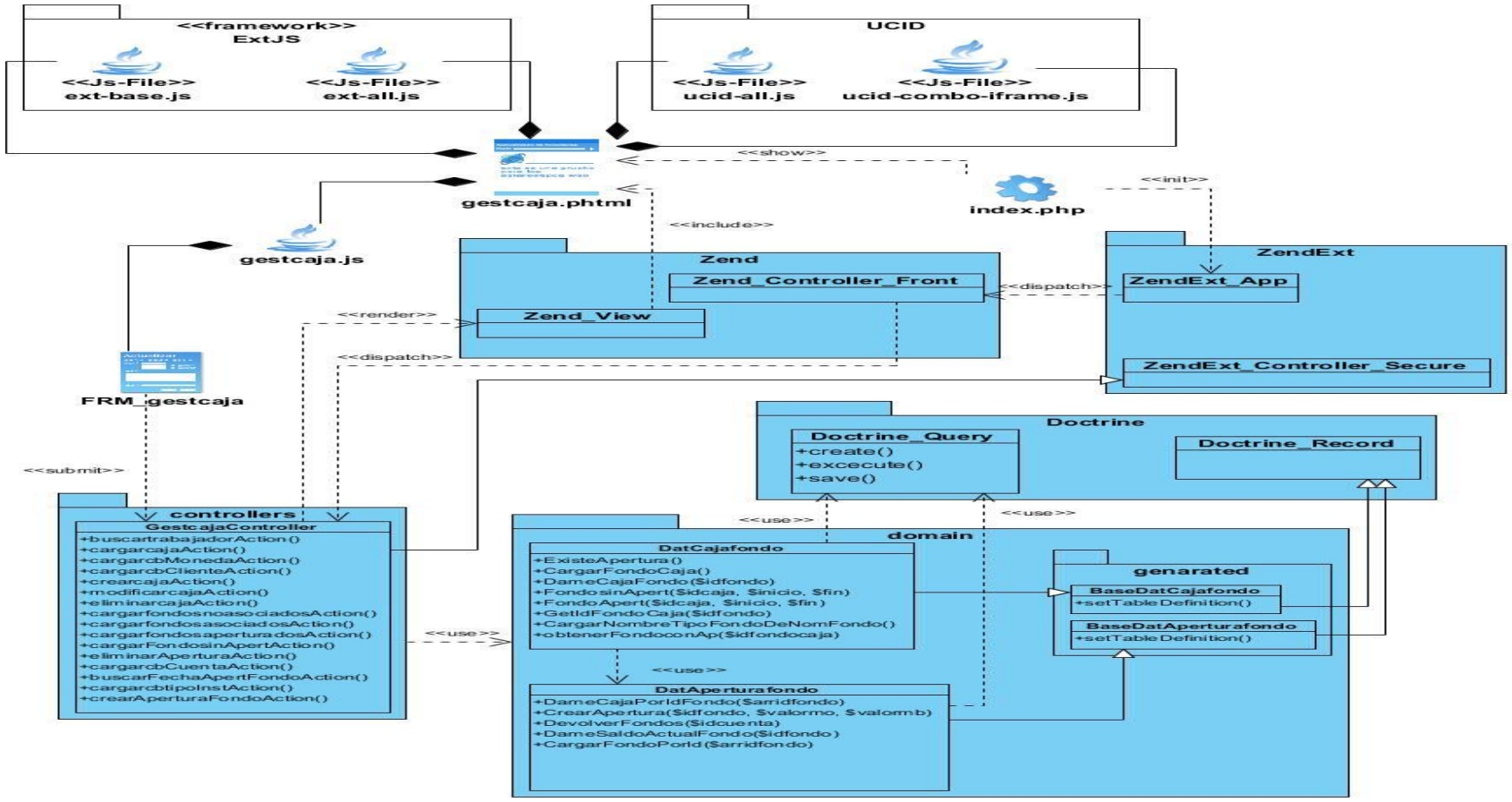

Figura 15.Diagrama de clases del diseño. Gestionar caja.

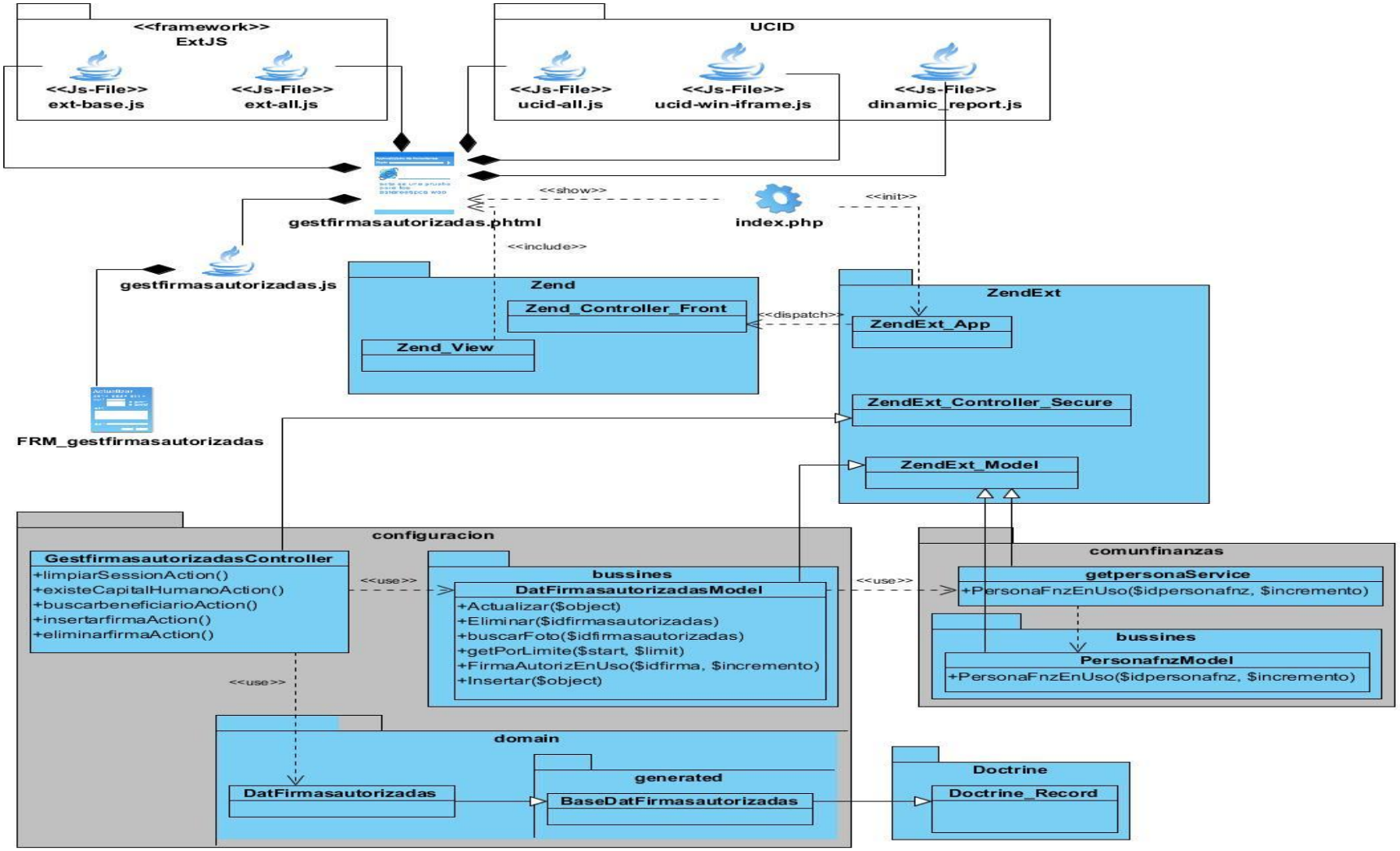

Figura 16. Diagrama de clases. Gestionar firmas autorizadas.

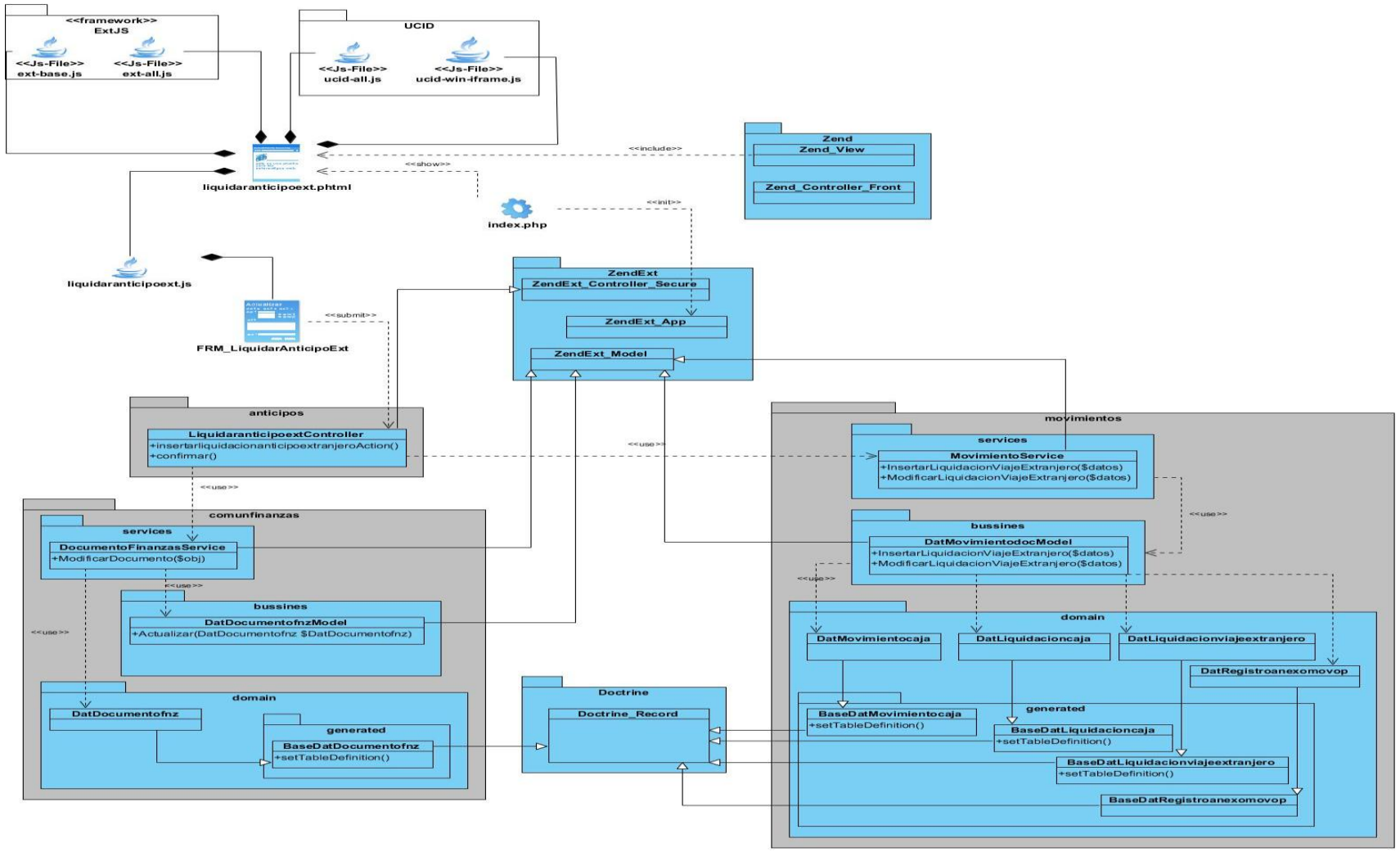

Figura 17. Diagrama de clases. Liquidar anticipos de dietas de gastos de viaje exterior.

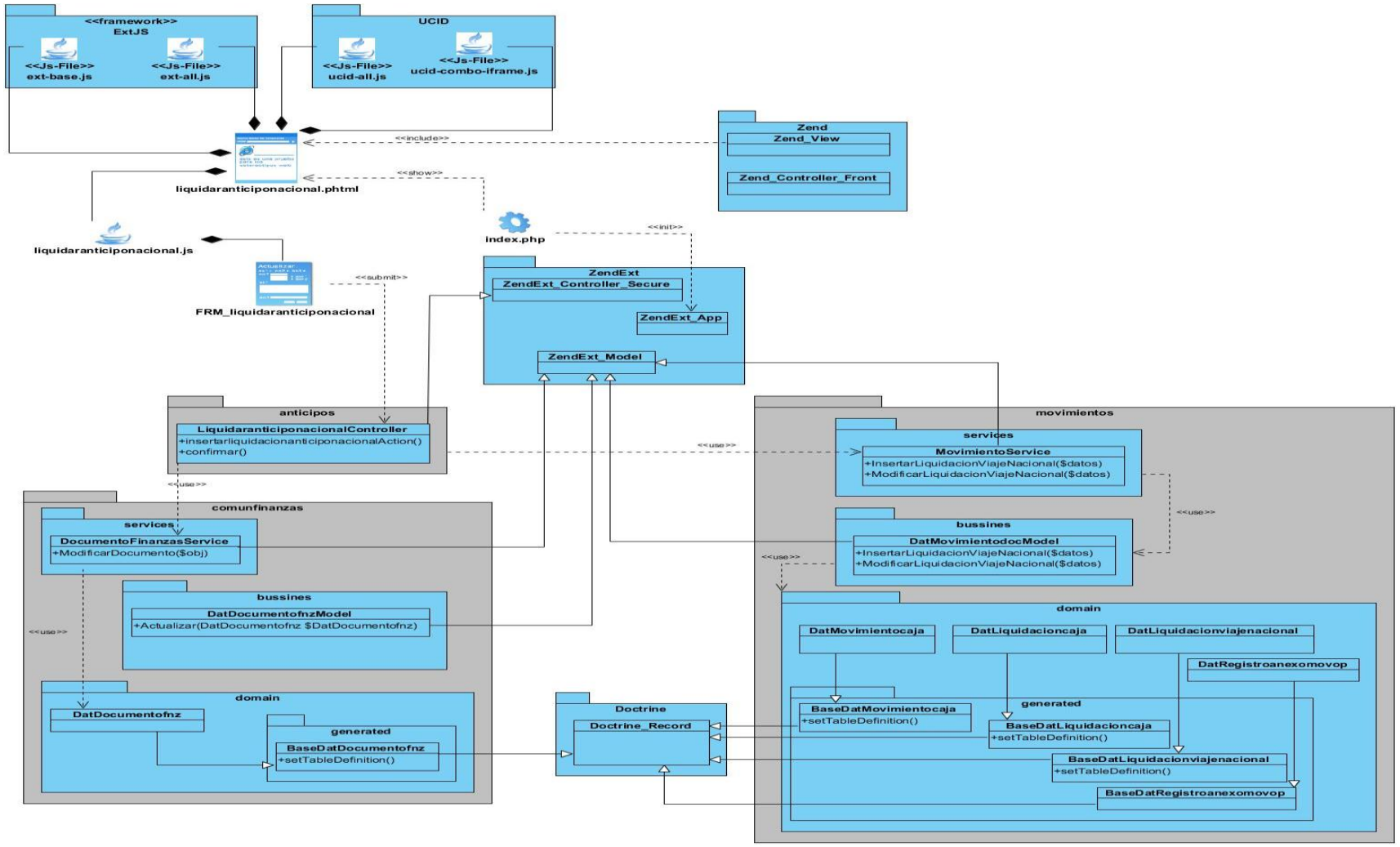

Figura 18. Diagrama de clases. Liquidar anticipos de dietas de gastos de viaje nacional.

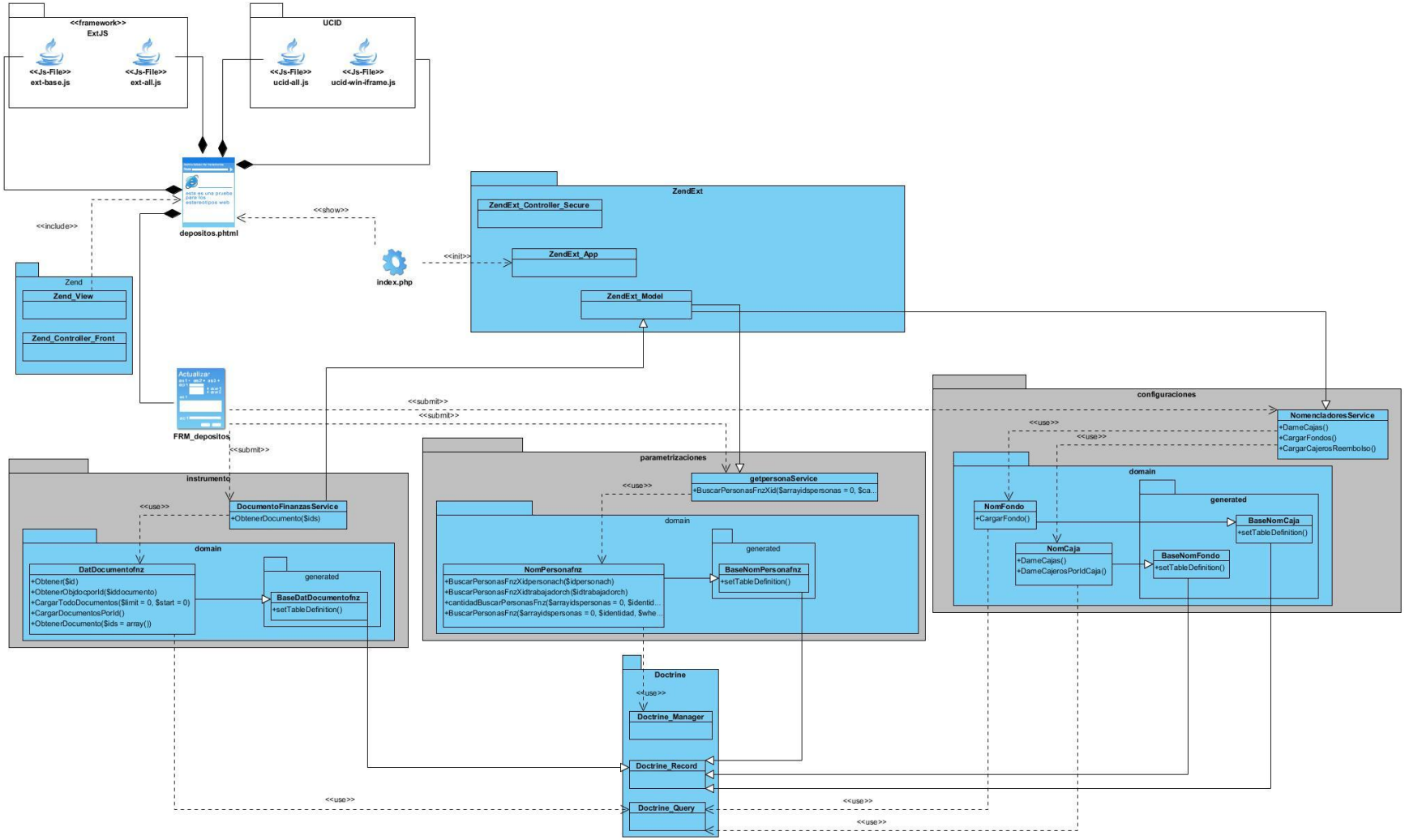

Figura 19. Diagrama de clases. Realizar Depósito.

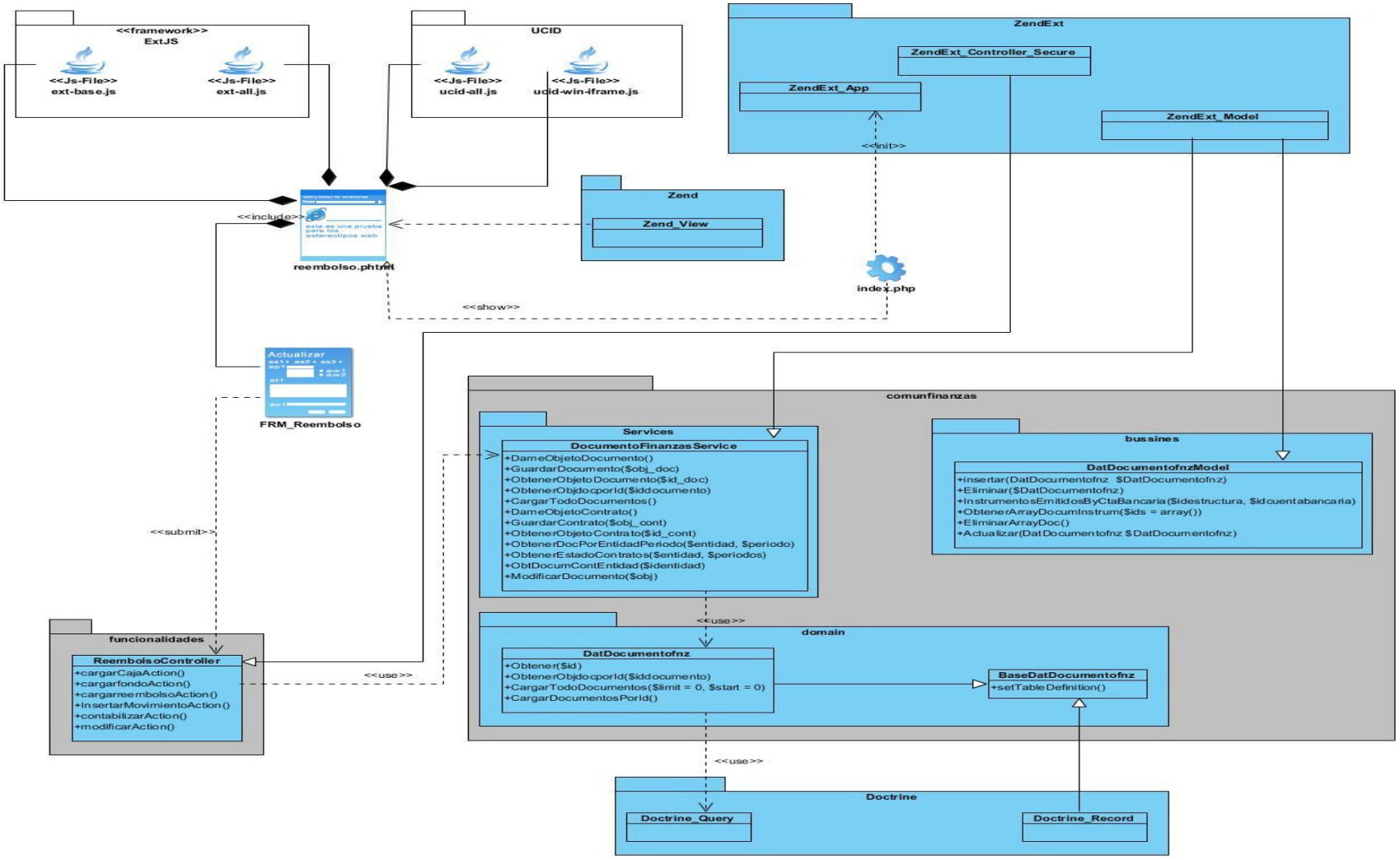

Figura 20. Diagrama de clases. Realizar Reembolso.

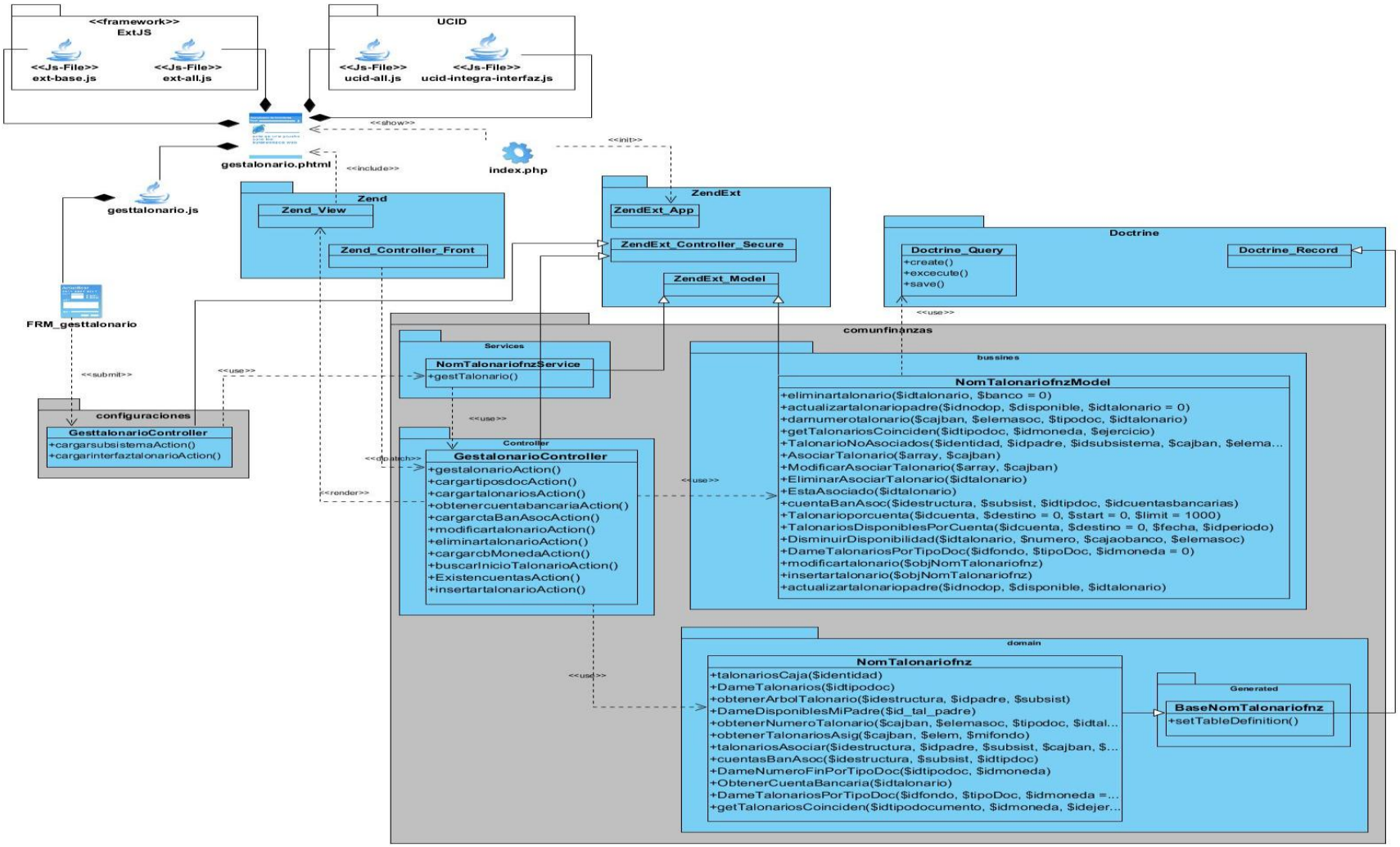

Figura 21. Diagrama de clases. Gestionar Talonario.

Anexo 2

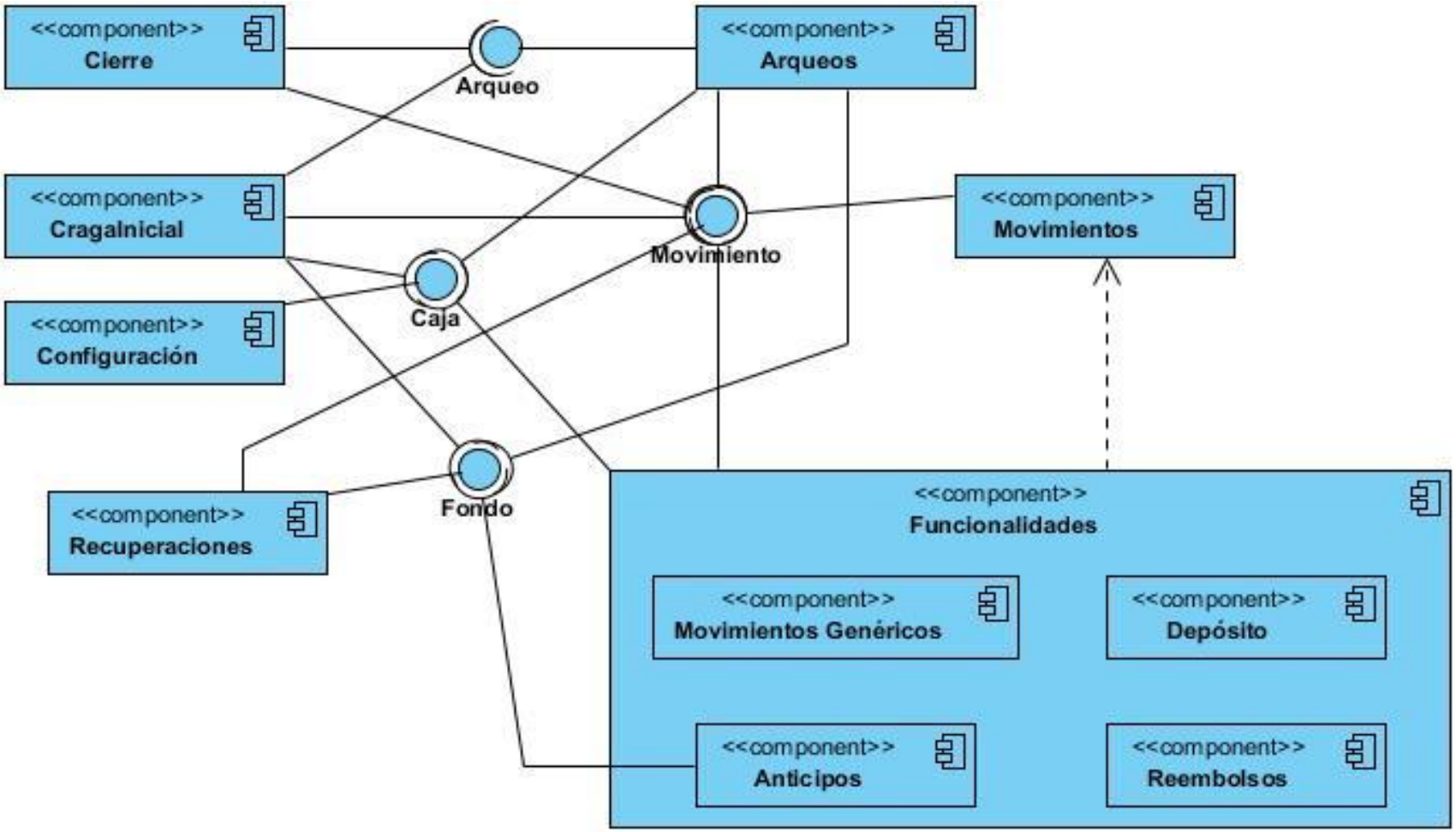

Figura 22. Diagrama de componentes. Módulo Caja

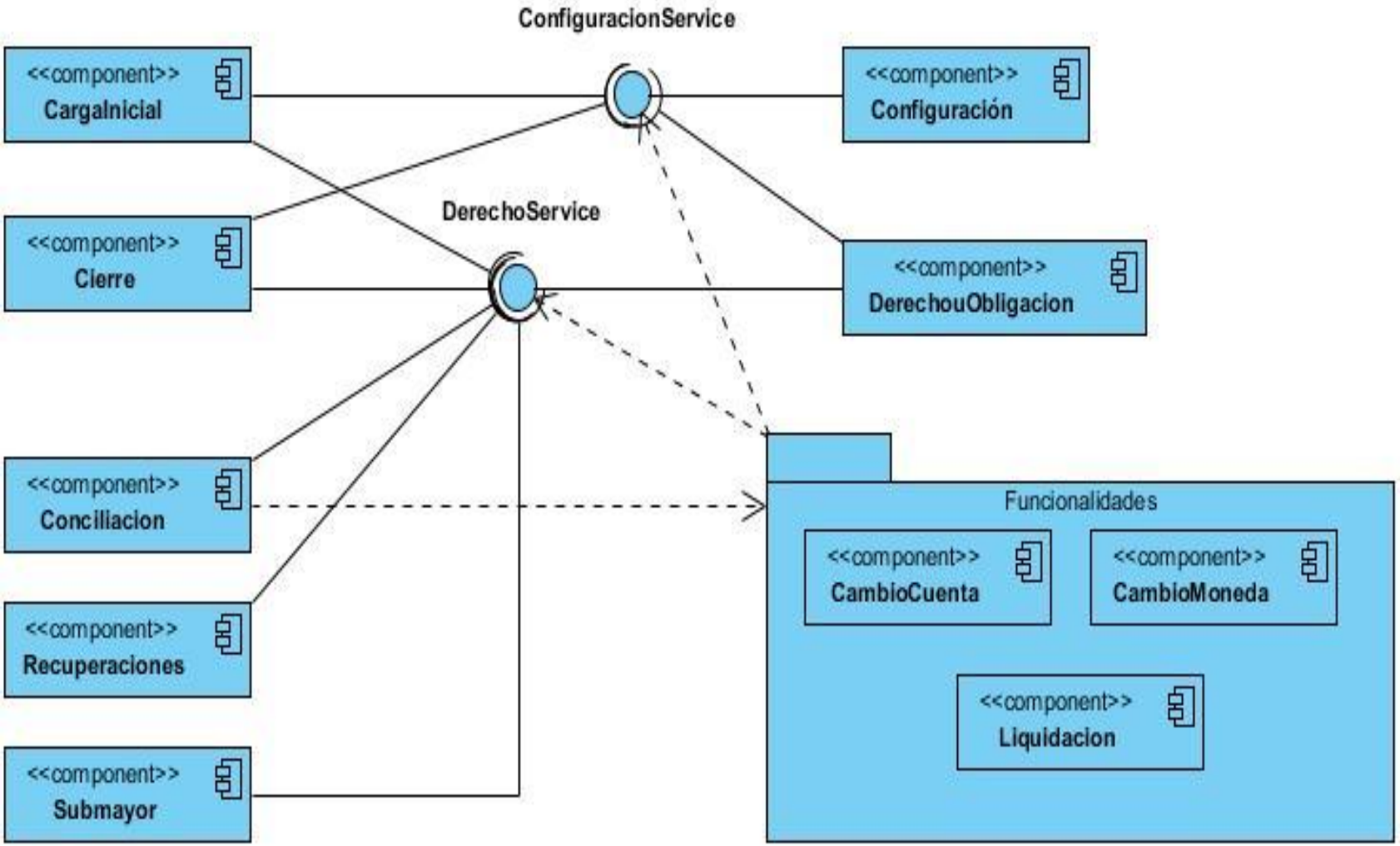

Figura 23. Diagrama de componentes. Módulo COPA