Universidad de las Ciencias Informáticas

Implementación del Módulo Cobros y Pagos del Sistema Integral de Gestión Cedrux.

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

Autores: Rafael Guevara Gutiérrez

Edimir Padilla Brooks

Tutores: Ing. Yanelis González de las Casas

Ing. Iyugnis Leyva Báez

Ciudad de la Habana, Junio 2009

DATOS DE CONTACTO

Ing. Yanelis González de las Casas

Correo electrónico: yanelisgc@uci.cu

Título de Graduado: Ingeniero en Ciencias Informáticas.

Ing. Iyugnis Leyva Báez

Correo electrónico: ileyva@uci.cu

Título de Graduado: Ingeniero en Ciencias Informáticas.

"La ciencia no es ni misterio de iniciados, ni privilegio de los aristócratas de la mente, sino el medio único que tiene el hombre de explicarse las leyes de la vida"

José Martí

AGRADECIMIENTOS

*A mis **padres** y a mi **familia** por la educación y el apoyo incondicional que me han dado durante todos estos años.*

*A la **Revolución** por haberme dado la posibilidad de estudiar, por engendrar tantos sueños y hacer el mío realidad.*

*A **Fidel** por ser siempre el guía que nos ha sabido conducir por el camino correcto en los buenos y malos momentos, por sus brillantes ideas, de las cuales nació esta universidad.*

*A la **UCI** que me permitió forjarme como profesional y como ser humano.*

*A todos los **profesores** que han contribuido de una forma u otra en mi formación profesional.*

*A mis **tutoras** Yanelis e Iyugnis.*

*A todos mis **amigos**, en especial a Yasmany Pino, por confiar en mi amistad y brindarme su apoyo desinteresado. Nos los olvidaré nunca.*

*A mis **compañeros** del Proyecto ERP-Cuba con los que he compartido alegres momentos, pero también largas y duras jornadas que me enseñaron lo que es el verdadero trabajo en equipo.*

*A mi compañero de tesis **Edimir** y a todas las personas que han puesto su granito de arena en la realización de este trabajo.*

A todos, muchas gracias.

Rafael

*A mis **padres** por la educación, el tiempo dedicado, su amor, la confianza y por haberme dado la vida, muchas gracias los quiero mucho. A mi hermano **Carlos Alberto Lacre Brooks** por estar siempre apoyándome y siguiendo mis pasos; mi hermano, el más alegre, mi hermanito, tú me has traído alegría desde el día que naciste. A mi **abuela**, mis **tíos** y **tías**, en especial **Mariela** por todo lo que ha hecho por mí. A mi **familia** en general por todo su apoyo y cariño. A mi amigo **Néstor Bernal Vidal**, un amigo que quiero como a un hermano, el cual ha estado a mi lado y hemos compartido todos esos secretos y aventuras que solo se pueden vivir entre amigos verdaderos he incondicionales. A mi hermana **Gretter**, la más chiquita, que prácticamente le ha tocado vivir una historia semejante a la mía con mi padre. A **Wendy Gracia Valdés** que se encuentra cumpliendo misión, gracias por brindarme tu amistad. A **René Lazo** por aceptarme en el proyecto ERP, por ser ejemplo, por su esfuerzo, sus conocimientos, gracias por todo. A los profesores que me han apoyado durante mi transcurso por la Universidad. A mis tutoras **Iyugnis** y **Yanelis** por su dedicación y comprensión en todo momento gracias por lograr este sueño. A mis nuevos e inolvidables amigos **Rigoberto "El Grande"** y **Walberto "El uno"**, que me ayudaron mucho para que este trabajo tuviese la funcionalidad y calidad requerida, muchas gracias. A mi compañero de tesis **Rafael** y a todos aquellos que de una forma u otra ayudaron en mi formación y en la realización de este trabajo.*

Edimir

DEDICATORIA

*A mi mamá **Iraida** y a mi papá **Rafael**, por estar siempre ahí y quererme tanto.*

*A toda mi **familia**, los que están y los que no, por ayudarme a ser la persona que soy hoy.*

*A la **Revolución** y a **Fidel** por darme la oportunidad de cumplir un sueño.*

*A todos mis **amigos** y **amigas**, los de aquí y los de Santa Clara.*

*A mis **compañeros** de estudio y trabajo durante estos cinco años.*

A todos dedico este momento porque de todos es.

Rafael

*A mi mamá **Maritza Brooks García** por ser siempre mi guía, mi lamparita mágica, mi espejo, mi ejemplo a seguir, por ser quien ha estado a mi lado en todo momento dándome las fuerzas necesarias para continuar luchando día tras día y seguir adelante rompiendo todas las barreras que se me han presentado, gracias a ella soy quien soy hoy en día, fue la que me dio ese cariño y calor humano necesario, la que veló por mi salud, mis estudios, mi educación , alimentación, entre muchas otras cosas, es ella a quien le debo todo; horas de consejos , de regaños , de reprimidas, de tristezas y de alegrías de las cuales estoy muy seguro que las ha hecho con todo el amor del mundo para formarme como un ser integral y de las cuales me siento extremadamente orgulloso; madre, Ud. más que nadie sabe por todo lo que hemos pasado durante mis estudios en la Universidad, por eso es a Ud. a quien le dedico principalmente esta tesis y el título que está por llegar, ma gracias por existir y estar siempre a mi lado. A mi **hermano**, espero que te sirva de ejemplo e inspiración estoy seguro que pronto lo lograrás. Dedico este proyecto de Tesis a mi carrera.*

Edimir

RESUMEN

A partir de la década de los 60, con la introducción de la computación en el mundo, se comenzó a generar una demanda de software para dar solución a disímiles problemas que se presentaban en la época. En los últimos años se ha venido experimentando un acelerado avance de las tecnologías de la información y las comunicaciones (TIC); nuestro país no se encuentra ajeno a este proceso, y dado el elevado volumen de información que se genera actualmente en las entidades cubanas es que las mismas han tratado de aplicar y extender su uso.

A pesar de la existencia de varios software tanto nacionales como extranjeros para la gestión de los procesos contables de nuestras entidades, se hace evidente que su desempeño no resulta ser el mejor, ya que no todos se adecuan a las particularidades de las instituciones del país; es por ello que se dificulta lograr una aplicación estándar que se adapte a necesidades específicas de cada tipo de entidad, haciéndose necesario un software que controle los procesos empresariales, como son los sistemas de Planificación de Recursos Empresariales (ERP), adaptados a las características propias del país.

El presente trabajo de diploma contribuirá a agilizar los procesos de gestión económica para las entidades del país, a través de la implementación del módulo contable “Cobros y Pagos” (COPA) del Sistema Integral de Gestión Cedrux, cumpliendo con los requerimientos analizados y priorizados por los usuarios funcionales del mismo e integrado con otros subsistemas que son necesarios para su correcto funcionamiento y flujo de información; permitiéndoles un mayor control de sus recursos y aumentar la eficiencia en las operaciones contables y cumpliendo con lo establecido.

PALABRAS CLAVE

Cobros, Pagos, ERP.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La contabilidad es una técnica en constante evolución y se encarga de registrar, clasificar y resumir en términos financieros las operaciones económicas, por medio de ella y una administración financiera eficiente se interpretan los resultados obtenidos y se proyecta el desempeño futuro de una entidad, lo que representa un medio efectivo para la dirección, el control de los recursos y su utilización eficiente.

Entre los principales objetivos de la Contabilidad podemos encontrar:

- ✓ Registrar en forma metodológica las operaciones de carácter financiero que ocurren en una empresa.
- ✓ Suministrar información clara y precisa de la situación financiera de una empresa en un momento determinado, así como de los resultados de las operaciones en un período delimitado.
- ✓ Analizar e interpretar los resultados obtenidos en la actividad de una empresa.
- ✓ Sirve para elaborar presupuestos de diversas índoles sobre la actividad futura de la empresa.
- ✓ Presenta datos precisos, medibles y analizables que le permiten a la administración tomar decisiones en cualquier momento, establecer responsabilidades, definir políticas, delegar autoridad, etc.

Dada su elevada importancia, la contabilidad es una técnica mundialmente utilizada, y las instituciones cubanas hacen uso de ella ya sea para su actividad presupuestada o empresarial según el caso.

Aparejado a esto, el creciente desarrollo informático ha hecho de esta ciencia prácticamente indispensable para el desarrollo de los procesos empresariales dado lo cambiante del mercado y los cada vez más complejos negocios que requieren grandes volúmenes de información. Esto ha llevado a que las entidades precisen del uso de aplicaciones como son los sistemas ERP, que son aplicaciones estructuradas que satisfacen la demanda de soluciones de gestión empresarial, y de forma general las expectativas del cliente, unificando las áreas y departamentos de las entidades, haciendo posible una administración eficiente, maximizando beneficios y minimizando costos.

En el mundo existen varios ERP que son utilizados por múltiples empresas, algunos de los cuales poseen el módulo de Cobros y Pagos, pero dado que los mismos han sido diseñados para empresas capitalistas su desempeño se ve afectado, puesto que estas cuentan con modelos de gestión y procesos diferentes al de las entidades

cubanas. No por esto se hacen totalmente inutilizables, pero el proceso de adaptarlos a las necesidades de las entidades del país se hace tan engorroso que sería preferible diseñar nuestro propio sistema de acuerdo a nuestras características propias.

Por otro lado las tecnologías utilizadas por estos sistemas son muy costosas, ya que utilizan paquetes de software propietarios que implican el pago de licencias, en muchos casos imposibles de obtener dadas las restricciones que nos impone el bloqueo económico, financiero y comercial del gobierno de Estados Unidos.

En la actualidad algunas entidades del país disponen de sistemas de gestión de procesos contables de cobros y pagos, pero los mismos no se ajustan del todo a sus necesidades reales. En otras entidades no se dispone de dichos sistemas implicando que todo el proceso se realice manualmente, archivándose lo relacionado con ellos en papel, lo cual representa un riesgo para el normal y correcto desempeño de la gestión contable de la entidad, ya que la probabilidad de cometer errores al realizar este proceso manualmente es muy alta, impidiendo que se realice de forma eficiente el proceso de toma de decisiones en la entidad.

Partiendo de lo anteriormente expuesto, se nos plantea el siguiente **problema:** Los funcionarios encargados de realizar los procesos de cobros y pagos en las entidades cubanas no cuentan con una herramienta que englobe todas las funcionalidades necesarias, que les viabilice y facilite su trabajo.

El **objeto de estudio** del presente trabajo está enmarcado en los procesos contables de cobros y pagos en las entidades del país, el **campo de acción** se centra en la implementación de sistemas que gestionen los procesos de cobros y pagos en las entidades cubanas.

El **objetivo general** del presente trabajo es informatizar el módulo Cobros y Pagos en correspondencia con los requerimientos de los usuarios, utilizando un modelo de desarrollo y lenguaje que responda a las nuevas concepciones de informatización en el país y que permita una realización eficiente de los procesos que aquí se llevan a cabo.

Para realizar con éxito el presente trabajo y cumplir con el objetivo propuesto, han sido definidas las siguientes **tareas:**

- ✓ Confeccionar el marco teórico-conceptual de la investigación a partir de una búsqueda y revisión bibliográfica.
- ✓ Realizar un estudio de los artefactos entregados por los analistas del módulo.
- ✓ Estudiar el marco de trabajo desarrollado por Línea de Arquitectura del ERP.
- ✓ Estudiar los estándares de codificación definidos por la Línea de Arquitectura.

- ✓ Implementar el módulo de Cobros y Pagos para lograr la informatización de sus procesos.

Para llevar a cabo este trabajo se ha planteado la siguiente **idea a defender:** si se lleva a cabo la implementación del módulo Cobros y Pagos, entonces existirá una herramienta que automatice las labores y contribuya a una rápida y eficiente gestión de los procesos de cobros y pagos.

Se utilizaron diferentes **métodos de investigación** tales como.

- ✓ Analítico-Sintético: análisis de documentos con el objetivo de determinar los puntos esenciales del proceso.
- ✓ Histórico-Lógico: Para determinar si actualmente están desarrollados sistemas informáticos similares.

Como **posible resultado** se espera obtener un módulo del subsistema de Finanzas del Sistema Integral de Gestión Cedrux totalmente funcional, que cumpla con los requerimientos analizados y priorizados por los usuarios funcionales del mismo e integrado con otros subsistemas que son necesarios para su correcto funcionamiento y flujo de información. Además permitirá realizar todas las operaciones contables de una forma ágil y cumpliendo todo lo establecido.

El documento está estructurado en tres capítulos.

CAPÍTULO 1: Contiene la fundamentación teórica del tema tratado en la investigación. Se hace una descripción de las tendencias y tecnologías actuales que se utilizaron como soporte de la propuesta.

CAPÍTULO 2: Se realiza un análisis de los artefactos de la ingeniería de software que fueron entregados por los analistas. Se hace una descripción de los algoritmos no triviales implementados, así como un análisis de las principales estructuras de datos usadas, y finalmente se hace una descripción de las principales clases que se utilizaron.

CAPÍTULO 3: Se aplican métricas para evaluar el diseño propuesto y se hace un análisis de los casos de prueba que se aplicaron al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se definen conceptos importantes que ayudarán al entendimiento del módulo a desarrollar, se aborda también información referente a algunos sistemas informáticos vinculados a este campo de acción como son: Openbravo, SAP, Versat – Sarasola, SABIC, entre otros. Se hace además un análisis justificando las herramientas y tecnologías a utilizar en el desarrollo del módulo que se pretende implementar.

1.2 Sistemas de Planificación de Recursos Empresariales.

Un ERP es un sistema integral de gestión empresarial que está diseñado para modelar e informatizar la mayoría de los procesos en una entidad (área de finanzas, comercial, logística, producción). Su misión es facilitar la planificación de todos los recursos de la entidad.

Lo más destacable de un ERP es que agrupa y ordena toda la información de la entidad en un solo lugar, de este modo cualquier suceso queda a la vista de forma inmediata, posibilitando la toma de decisiones de forma más rápida y segura, acortando los ciclos productivos. [1]

1.2.1 Áreas implicadas

Un sistema ERP debe estar compuesto por un conjunto básico de módulos. Como mínimo tiene que ofrecer soluciones de gestión de la producción, financieras, de compras, ventas y logística, satisfaciendo las necesidades de las empresas en esos aspectos; además puede contener otros módulos como pueden ser los de gestión de recursos humanos y de proyectos.

1.2.2 Características principales

Existen tres características esenciales que distinguen a un ERP y es que son sistemas integrales, modulares y adaptables:

- ✓ Integrales, ya que permiten controlar los diferentes procesos de la empresa entendiendo que todos los departamentos de la misma se relacionan entre si, es decir, que el resultado de un proceso es punto de inicio del siguiente.
- ✓ Modulares, porque los ERP entienden que una empresa es un conjunto de departamentos que se encuentran interrelacionados por la información que comparten y que se genera a partir de sus procesos. Una ventaja de estos

sistemas, tanto económica como técnicamente es que se encuentra dividido en módulos, los cuales pueden instalarse de acuerdo con los requerimientos del cliente.

- ✓ Adaptables, ya que los ERP están creados para ajustarse a las características propias de cada empresa. Esto se logra por medio de la configuración o parametrización de los procesos de acuerdo con las salidas que se necesiten de cada uno. [2]

1.2.3 Algunas ventajas de los Sistemas de Planificación de Recursos Empresariales.

Varios son los puntos de vista en cuanto a los diferentes beneficios que se esperan en una implementación de un ERP, así como los impactos que este tendrá en la organización.

Entre las ventajas que otorgan estos sistemas se encuentran:

- ✓ Aumento de la productividad.
- ✓ Reducción de inventarios.
- ✓ Incremento de las ventas por tiempo de respuesta a clientes.
- ✓ Disminución de compras.
- ✓ Disminución de comisiones bancarias por cheques expedidos por órdenes.[3]

1.3 Subsistema Contable de Cobros y Pagos.

El subsistema de Cobros y Pagos incluye los procesos contables: cuentas por cobrar y cuentas por pagar.

Cuentas por cobrar: Son todas aquellas cuentas que se habilitan por concesiones que realiza el vendedor al comprador teniendo en cuentas las condiciones de este último. El tiempo de las cuentas por cobrar y su monto deberá estar en correspondencia con las características del comprador, las que conocerá el vendedor realizando entre otros los siguientes pasos:

- ✓ Conocer el valor crediticio de la entidad.
- ✓ Prestigio en el negocio.
- ✓ Permanencia de los ejecutivos.
- ✓ Capacidad financiera de acuerdo al tipo de negocio.
- ✓ Informe de comisiones evaluadora de créditos.
- ✓ Certificación de Estados Financieros por auditores reconocidos.
- ✓ Posición en el mercado.

- ✓ Comportamiento de ventas anteriores.

Cuentas por pagar: Son las deudas u obligaciones contraídas por la entidad por servicios recibidos o mercancías compradas al crédito, la cual está amparada generalmente por una factura, y la cual se debe cancelar, generalmente, en un lapso menor de un año.

1.3.1 Actividades de control vinculadas con este subsistema.

Cuentas por Cobrar:

- ✓ Custodia y archivo de los modelos factura en blanco y su numeración consecutiva.
- ✓ Custodia y archivo del control de las facturas canceladas y definición de la(s) causa(s).
- ✓ Confirmación de cobros con clientes seleccionados, según lo establecido en la legislación vigente.
- ✓ Las facturas comerciales emitidas se corresponden con los artículos enviados y recepcionados por el cliente.
- ✓ Registro oportuno de las facturas comerciales, cuyos envíos estén autorizados por la autoridad facultada.
- ✓ Registro exacto de las devoluciones de ventas autorizadas por la autoridad facultada.
- ✓ Los saldos que muestran las cuentas por cobrar están debidamente sustentados por las facturas comerciales realmente enviadas al cliente.
- ✓ Cobros anticipados debidamente controlados y sustentados por los documentos correspondientes.
- ✓ Custodia y archivo correcto de los expedientes de clientes, así como evidencia documental de las gestiones de cobro.
- ✓ Tratamiento de los faltantes, pérdidas y sobrantes, de acuerdo con la legislación vigente.
- ✓ Las tareas y responsabilidades relativas a las transacciones y hechos deben estar autorizadas, registradas y revisadas por personas diferentes, en los casos debidamente justificados.

Cuentas por Pagar:

- ✓ Confirmación de pagos con proveedores seleccionados, según lo establecido en la legislación vigente.
- ✓ Registro exacto y oportuno de las facturas de compras debidamente aprobadas por la autoridad facultada.

- ✓ Registro correcto de las devoluciones y bonificaciones recibidas de los créditos comerciales aprobados por la autoridad facultada.
- ✓ Los saldos que muestran las cuentas por pagar están debidamente sustentados por las facturas comerciales realmente recibidas del proveedor.
- ✓ Pagos anticipados debidamente controlados y sustentados por los documentos correspondientes.
- ✓ Tratamiento aplicado a las reclamaciones a los proveedores por mercancías no recibidas.
- ✓ Custodia y archivo correcto de los expedientes de proveedores.
- ✓ Tratamiento de los faltantes, pérdidas y sobrantes, de acuerdo con la legislación vigente.
- ✓ Las tareas y responsabilidades relativas a las transacciones y hechos deben estar autorizadas, registradas y revisadas por personas diferentes, en los casos debidamente justificados.

1.3.2 Importancia de la informatización de los procesos contables de cobros y pagos.

Los procesos contables de cobros y pagos tienen gran importancia para el funcionamiento de cualquier entidad, por tanto son un punto de obligatoria referencia a la hora de relacionar la entidad con sus clientes y proveedores. En la actualidad muchas entidades están obligadas a realizar estos procesos de forma manual, esto se convierte en una tarea muy difícil a la hora de llevar a cabo un buen control interno.

La realización de un sistema informático que permita gestionar los procesos de cobros y pagos, trae consigo mejoras considerables para las entidades y sus clientes a la hora de relacionarse y llevar a cabo estos procesos, disminuyendo en gran medida la ocurrencia de errores humanos, hechos delictivos, y la información fluye con claridad y mayor precisión permitiendo un sistema contable más seguro. La informática aplicada a estos procesos contables hace que el Contador pueda igualmente desarrollar una mejor actividad profesional, al considerar esa técnica como un medio también de agilizar y facilitar su trabajo.

1.4 Sistemas usados para la gestión contable

Con el vertiginoso desarrollo de la informática a escala global ha surgido de necesidad de automatizar disímiles esferas de la economía, con el fin de mitigar errores que el ser humano comete sin darse cuenta y acelerar el trabajo. Es por ello que muchas empresas en los ámbitos internacional y nacional emplean sistemas informáticos

dedicados a llevar los procesos contables de manera eficiente y solo con los recursos humanos necesarios.
Seguidamente se ofrecen algunos ejemplos de estos sistemas.

- SABIC:

El SABIC (Sistema automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de Bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado.
Este sistema ha sido adaptado a los requerimientos de las operaciones propias del Banco Central y ha sido desarrollado para que los empleados que hagan uso de él puedan tramitar sus operaciones y realizar sus consultas sin necesidad de acudir a los archivos ni a la actividad manual. De esta forma se aumenta la seguridad, la eficiencia del trabajo y la productividad de los trabajadores.
Entre sus principales características, está la contabilización en tiempo real (que permite mantener actualizados los ficheros contables) y contabilización multimoneda (que permite registrar los activos y pasivos en las monedas orígenes sin tener que realizar en el momento del registro las conversiones de monedas, lo cual aumenta la exactitud de la información sobre la posición financiera de la institución). Además, las operaciones contables se pueden realizar a través de transacciones tipificadas que generan los asientos contables de forma automática.
Teniendo en cuenta lo dinámico que resulta la actividad bancaria, este sistema está concebido modularmente: Un módulo central y módulos de transacciones, listados y procesos.
El módulo central es uniforme para todas las aplicaciones, se encarga del control de accesos, actualización de ficheros y los procesos de inicio y cierre del día contable. Los módulos restantes son adaptados a los requerimientos de la actividad específica de cada institución, siendo el módulo de transacciones el que se encarga de dar una entrada uniforme y coherente de los datos de la contabilidad en el sistema y es el que identifica las operaciones de cada entidad.
Esta modularidad tiene la intención de facilitar la adaptabilidad y el mantenimiento del sistema para garantizar su evolución; o sea, se puede modificar cualquier módulo sin que los demás se vean afectados. [6]

- Versat – Sarasola:

Este es un software creado por el Licenciado en Economía y villaclareño Miguel Cabrera González, este es un sistema de contabilidad confiable, ofrece mayor

organización, control y disciplina en cada gestión, posibilita enviar información eficaz, de forma inmediata y desde lugares apartados. Este proyecto resulta ser un sistema integrado, constituido por 12 módulos que incluyen:

- ✓ Configuración y seguridad
- ✓ Contabilidad general y de gastos
- ✓ Costos y procesos
- ✓ Análisis económico empresarial
- ✓ Control de activos fijos
- ✓ Planificación y presupuestos
- ✓ Control de inventarios
- ✓ De productos terminados
- ✓ Pago de salario
- ✓ Paquete de gestión
- ✓ Contratación y facturación.
- ✓ Finanzas, caja y banco

Este producto le ha traído grandes beneficios al país, pues se ahorró un millón 186 mil dólares que costaban las licencias al evitarse la importación del sistema foráneo, más labor de consultoría, atenciones y otros gastos, además de que actualmente lo utilizan alrededor de 200 entidades de varias provincias y siguen aumentando los clientes. [7]

- ➢ ASSETS-NS:

Este programa se introdujo en Cuba en el año 1997 y desde entonces se le han hecho muchos cambios para adaptarlo a la realidad de la economía cubana, estos cambios acercan más el producto a las tendencias actuales de explotación de software cubano, también este producto se usa en diferentes países como México, República Dominicana y España. Este software brinda una serie de beneficios “pues permite controlar la inmensa mayoría de las transacciones de una empresa, tanto de control de inventarios como de finanzas, cobros y pagos, contabilidad, recursos humanos, nóminas, y además auditoría... al estar registradas todas estas operaciones permite un control más efectivo por parte de los ejecutivos para adoptar decisiones acertadas en torno a la labor de sus empresas”, según la opinión de Raúl Carnota; Jefe del Grupo de Instaladores de INFOMASTER, proveedor exclusivo de ASSETS en Cuba. ASSETS-NS es un Sistema de Gestión Integral que se monta en una plataforma de servidores SQL, en la actualidad existen muchos sistemas de este tipo pero este software ha sido uno de los mejores en las diferentes licitaciones en las que ha participado. Mundialmente es un software líder pues está hecho con tecnología de punta y respaldado por una plataforma de datos muy poderosa. [8]

- Openbravo:

Openbravo es un sistema de planificación de recursos empresariales (ERP) en software libre y basado íntegramente en web. Dispone de soporte para bases de datos PostgreSQL y Oracle. Se encuentra disponible en español y actualmente se preparan localizaciones en varios otros idiomas. No dispone de clientes de utilización que no sean a través de un navegador Web. Cuenta con un módulo de gestión de los procesos de cobros y pagos.

Openbravo es la solución profesional líder en el mercado de los ERP en código libre y entorno Web dirigido a pequeñas y medianas empresas. Basado en la filosofía del software libre y de un servicio de alta calidad, Openbravo ofrece un sistema ERP totalmente integrado y en entorno Web, adaptado a las necesidades de cada empresa, independientemente de su tamaño o su sector de actividad. Las funcionalidades de Openbravo se encuentran en continuo crecimiento gracias a la constante expansión de su comunidad internacional de usuarios, y desarrolladores.

Openbravo ayuda a las empresas a administrar sus operaciones diarias, optimizar los procesos de negocios, lograr una mayor satisfacción del cliente y, en definitiva, incrementar su rentabilidad. Y debido a su característica de software libre, el cliente tiene el control completo de la solución, sin depender de contratos o licencias. [4]

Luego de realizado un análisis de varios sistemas de Cobros y Pagos, tanto nacionales como extranjeros se arribó a la conclusión de que pese a existir sistemas muy completos y funcionales que resuelven los procesos; estos son en su gran mayoría software privativos cuyas licencias provocan pérdidas millonarias al país en un año fiscal. También en el estudio realizado se concluyó que los sistemas de Cobros y Pagos extranjeros presentaban problemas de compatibilidad con el sistema financiero cubano.

Entre los sistemas nacionales estudiados se constató que en su mayoría no son multiplataforma y sus gestores de base de datos son privativos. Luego de este análisis de los sistemas de Cobros y Pagos estudiados se tomaron las mejores soluciones de ellos, para adaptarlas al sistema financiero cubano.

1.5 Tendencias y tecnologías actuales

Para implementar un software se hace necesario tener en cuenta una serie de tecnologías y herramientas disponibles para llevar a cabo dicha tarea. La decisión de cuales son las más idóneas a utilizar dependen de las características de la entidad donde se va a implantar el sistema que se obtenga, así como de lo que se desea del producto final unido a las ventajas que ofrece cada tecnología. Teniendo como

premisas lo antes expuesto seguidamente se realizará un análisis de las herramientas y tecnologías a utilizar en el desarrollo del producto final.

1.5.1 Metodología de Desarrollo

Todo desarrollo de software es riesgoso y difícil de controlar, si no se utiliza una metodología que guíe el proceso, se obtienen clientes insatisfechos con los resultados y desarrolladores aún más insatisfechos. El uso de una metodología garantiza determinadas características de gran importancia en los sistemas, dentro de ellas la calidad, es el factor primordial tanto para el cliente como para los desarrolladores, por otro lado esta el tiempo que es un factor crítico que afecta todo producto.

1.5.1.1 Modelo de desarrollo orientado a componentes.

El ERP es un sistema integral de gestión empresarial que está diseñado para modelar y automatizar la mayoría de procesos en la empresa (Contabilidad General, Caja, Banco, Costos y Procesos, Cobros y Pagos, Inventario, Auditoría, Planificación, etc.). Su misión es facilitar la planificación de todos los recursos de la empresa. El software ERP planea y automatiza muchos procesos con la meta de integrar información a lo largo de la empresa y elimina los complejos enlaces entre los sistemas de las diferentes áreas del negocio. Es un proyecto que por su complejidad en los procesos de negocio, tamaño, integración entre los subsistemas que lo conforman y además por el cúmulo de información que almacenan y analizan, requiere de una largo período de elaboración, construcción y de coordinación entre los equipos de desarrollo [9].

Para la exitosa realización de un proyecto de tales magnitudes es necesario establecer modelos estandarizados para los equipos inmersos en el desarrollo e implementación, así como una definición clara y precisa de las responsabilidades de los roles involucrados en la solución (ver Figura 1).

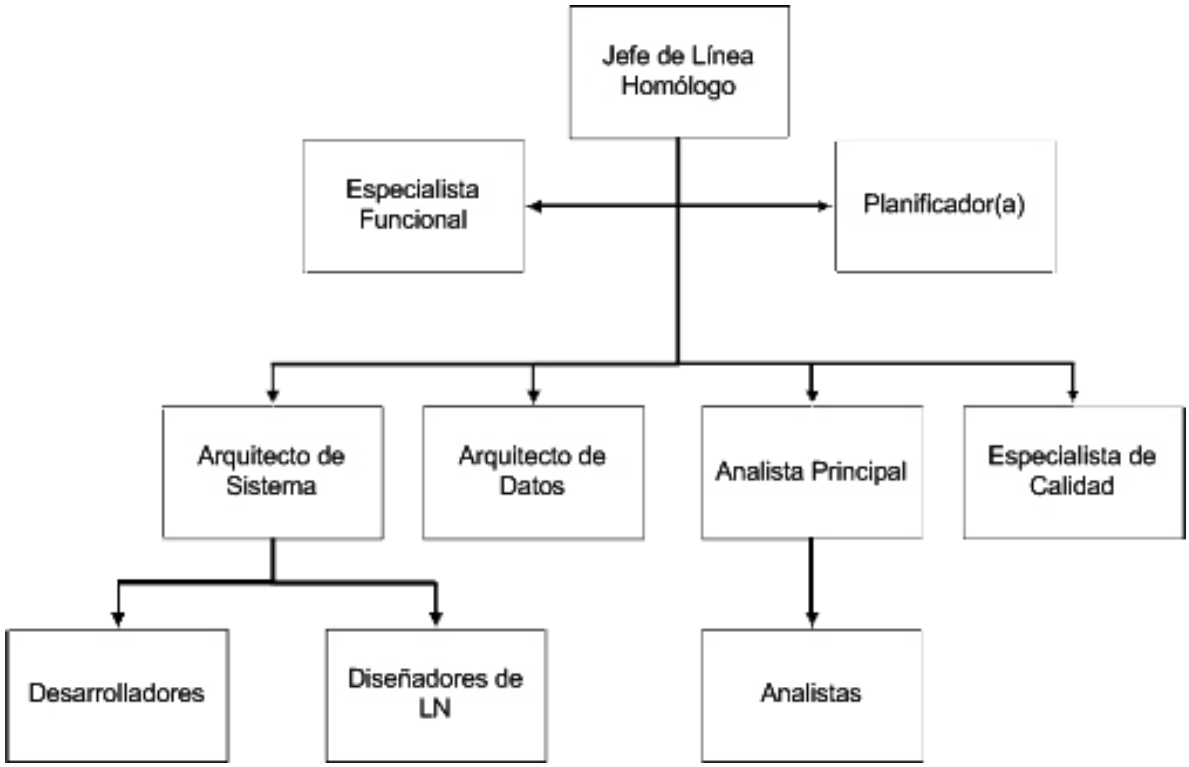


Figura 1 Organización de roles por línea.

De los roles mencionados en la figura anterior se empleó en la presente investigación el rol Implementador o desarrollador.

Rol de Desarrollador:

El desarrollador debe poseer habilidades de programación, y un gran conocimiento del marco de trabajo, así como estar familiarizado con las herramientas utilizadas en la implementación.

Responsabilidades:

- Diseña y Construye los componentes de software de la línea

Actividades en las que participa:

- Reunión de Implementación
- Implementación

Artefactos que genera:

- Plan de Trabajo Individual
- Implementación de componentes
- Descripción de los componentes

1.5.2 Aplicaciones Web

Las aplicaciones Web se desarrollan como una extensión de los Sistemas Web para agregar funcionalidades del negocio al proceso. En otros términos, una aplicación Web es un sistema Web que permite a los usuarios ejecutar lógica de negocio a través de un Navegador, o lo que es lo mismo: modificar el estado del negocio. Su arquitectura general es la de un sistema Cliente - Servidor. El protocolo principal de comunicación en una aplicación Web es HTTP, el cual funciona normalmente desconectado, es decir, el cliente hace una petición al servidor, este la procesa y le devuelve el resultado, terminando la comunicación entre estos. [11]

1.5.4 Framework

En general, un framework es una estructura real o conceptual que pretende servir como soporte o guía para la construcción de algo que expande su estructura en algo usable.

En sistemas computarizados, un framework es con frecuencia una estructura que indica cuales tipos de programas pueden o deben ser construidos y como estos se interrelacionaran. Algunos pueden además incluir programas existentes, interfaces de programación específicas u ofrecer herramientas de programación para usar el framework. Un framework puede ser para un conjunto de funciones dentro de un sistema y como ellos se interrelacionan, las capas de un sistema operativo, de un subsistema de aplicación, como la comunicación debe ser estandarizada en algún nivel de la red de trabajo. Es generalmente más comprensivo que un protocolo y más prescriptivo que una estructura.

1.5.5 IoC

IoC (Inversión of Control, IoC por sus siglas en inglés) es un concepto junto a técnicas de programación en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones. En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.

1.5.6 Lenguajes de programación para la Web

Los lenguajes de programación para la Web son aquellos que permiten que las aplicaciones sean dinámicas, posibilitando la interacción con el usuario y la personalización de la información, pueden ser del lado del servidor o del lado del cliente.

1.5.6.1 Lenguajes del lado del cliente

Los lenguajes del lado del cliente son aquellos que pueden ser directamente comprendidos por el navegador y no necesitan un pre-tratamiento, son totalmente independientes del servidor, lo cual permite que la página pueda ser albergada en cualquier sitio. Dentro de los lenguajes del lado del cliente se encuentran principalmente el JavaScript (JScript), el Visual Basic Script (VBScript) y el EXTjs, que son los encargados de aportar dinamismo a la aplicación en los navegadores. En el caso del VBScript éste es prácticamente usado a la hora de programar en ASP del lado del servidor, ya que su mayor desventaja radica en que solo es soportado por el navegador Web de su fabricante, Microsoft. Por otro lado EXTjs y JScript son soportados por la mayoría de los navegadores existentes actualmente. [12]

Para la programación del lado del cliente se utilizará EXTjs.

- ✓ **EXTjs**

EXTjs es un framework que permite hacer interfaces bastante poderosas y amigables para sistemas sobre internet, usando tecnologías como AJAX, DHTML y DOM. Entre sus características más interesante tenemos que es cross-browser, es decir no es necesario hacer diferentes programaciones para los múltiples navegadores. Entre sus cualidades están un alto rendimiento, interfaces personalizables y tiene un API muy intuitiva. EXTjs posee dos tipos de licencias, LGPL y comercial, con la adquisición de esta última, el usuario recibe soporte por parte del equipo de desarrolladores de EXTjs. [13]

- ✓ **HTML**

HyperText Markup Language (Lenguaje de Marcado de Hipertexto) es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web. Gracias a Internet y a navegadores como Explorer o Netscape, el HTML se ha convertido en uno de los formatos más populares que existen para la construcción de documentos. [14]

- ✓ **XML**

XML es una tecnología que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con unas posibilidades mucho

mayores. También conocida como un lenguaje universal de marcado para documentos estructurados y datos en la Web, más amplio, más rico y dinámico que HTML. No solo es un lenguaje de marcado, sino también un metalenguaje que permite describir otros lenguajes de marcado, además permite que los diseñadores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones. [15]

- ✓ **AJAX**

No es exactamente un lenguaje su nombre viene dado por las siglas de Asynchronous JavaScript y XML, es un término que describe un nuevo acercamiento a usar un conjunto de tecnologías existentes juntas, incluyendo las siguientes: HTML o XHTML, hojas de estilo (Cascading Style Sheets o css), Javascript, el DOM (Document Object Model), XML, y el objeto XMLHttpRequest que nos permite realizar una conexión al servidor, enviarle una petición y recibir la respuesta que procesaremos en nuestro código Javascript. Cuando se combinan estas tecnologías en el modelo Ajax, las aplicaciones funcionan mucho más rápido, ya que las interfaces de usuario se pueden actualizar por partes sin tener que actualizar toda la página completa. [16]

1.5.6.2 Lenguajes del lado del servidor

Los lenguajes del lado del Servidor se caracterizan por desarrollar la lógica del negocio dentro del servidor, además de ser los encargados del acceso a base de datos y tratamiento de la información. Por el auge que han tenido, los más sobresalientes son: PERL (Lenguaje Práctico para la Extracción e Informe), ASP (Servidor de paginas Activas de sus siglas en ingles Active Server Pages), PHP, JSP (Servidor de Paginas en Java de sus siglas en ingles JavaServer Pages). [12]

Como lenguaje para la programación de los procesos a informatizar se ha elegido el PHP.

- ✓ **PHP**

PHP (procesador de hipertexto) es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas Web dinámicas, similar al ASP de Microsoft o el JSP de Sun, embebido en páginas HTML y ejecutado en el servidor. [17]

La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML o XML.

Al ser un lenguaje libre dispone de una gran cantidad de características que lo convierten en la herramienta ideal para la creación de páginas web dinámicas:

- Soporte para una gran cantidad de bases de datos: MysSQL, PostgreSQL, Oracle, Sybase, entre otras.
- Integración con varias bibliotecas externas, permite generar documentos en PDF (documentos de Acrobat Reader), analizar código XML.
- Perceptiblemente más fácil de mantener y poner al día que el código desarrollado en otros lenguajes.
- Soportado por una gran comunidad de desarrolladores, como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y reparen rápidamente.
- El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.

1.5.7 Servidor de aplicaciones Web

- ✓ **Apache**

Apache es un software libre de código abierto que funciona sobre cualquier plataforma, como por ejemplo, Unix, Windows, Macintosh y otras. Entre sus características presenta mensajes de error altamente configurables y base de datos de autenticación y negociado de contenidos. Es utilizado comúnmente para sitios con páginas estáticas. [18]

Ventajas:

- Modular (módulos cargados dinámicamente)
- Funciona sobre muchas plataformas
- Extensible
- Popular
- Gratuito.
- Apoyo fuerte para proveedores de Servicios de Internet (ISP's).
- Amplias librerías disponibles.
- Código fuente seguro

1.5.8 Sistemas Gestores de Bases de Datos (SGBD)

Los Sistemas Gestores de Bases de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre las bases de datos y las aplicaciones que la utilizan, los cuales permiten incorporar una serie de funciones que posibilitan la definición de los registros, sus campos, sus relaciones, insertar, eliminar, modificar y consultar los datos. El propósito general de los sistemas de gestión de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de informaciones.

Entre los SGBD más utilizados en el mundo se encuentran Oracle, MySQL, Microsoft SQL Server, PostgreSQL, InterBase, entre otros. Todos estos presentan un enfoque relacional con un buen basamento matemático centrado en el Álgebra Relacional.

Estos sistemas presentan disímiles ventajas, entre las que se encuentran:

- Facilidad de manejo de grandes volumen de información.
- Gran velocidad.
- Independencia del tratamiento de información.
- Seguridad de la información (acceso a usuarios autorizados), protección de información, de modificaciones, inclusiones, consulta.
- No hay duplicidad de información, comprobación de información en el momento de introducir la misma.

- ✓ **PostgreSQL**

Es un servidor de base de datos relacional, libre. Ofrece soporte total para transacciones, disparadores, vistas, procedimientos almacenados, almacenamiento de objetos de gran tamaño. Se destaca en ejecutar consultas complejas, consultas sobre vistas, sub-consultas de gran tamaño. Permite la definición de tipos de datos personalizados e incluye un modelo de seguridad completo.
Como toda herramienta de software libre PostgresSQL ofrece entre otras ventajas la de contar con una gran comunidad de desarrollo en Internet, su código fuente está disponible sin costo alguno y algo muy importante es que dicha herramienta es multiplataforma, está disponible en casi cualquier sistema operativo. Además de presentar gran estabilidad y confiabilidad, en contraste a muchos sistemas de bases de datos comerciales, es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad. [19]

1.5.9 Navegador

Un navegador web o explorador web es una aplicación que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web. Se decidió utilizar como navegador para conectarse a la aplicación el Mozilla Firefox 1.5 o superior.

- ✓ **Mozilla Firefox**

Es un navegador, con interfaz gráfica de usuario desarrollado por la Corporación Mozilla. El programa es multiplataforma y está disponible en versiones para Microsoft Windows, Mac OS X y GNU/Linux. El código fuente de Firefox está disponible libremente bajo la triple licencia de Mozilla como un programa libre y de código abierto.

Firefox incorpora bloqueo de ventanas emergentes, navegación por pestañas, marcadores dinámicos, compatibilidad con estándares abiertos, y un mecanismo para añadir funciones mediante extensiones.

1.5.10 Herramientas de desarrollo

- ✓ **Zend Studio**

Zend Studio es una herramienta profesional excelente para trabajar proyectos PHP. Es concebido con el fin de crear aplicaciones altamente fiables, proporciona una facilidad de uso inigualable, escalabilidad, fiabilidad, y la extensión que los programadores profesionales y de empresas requieren para desarrollar, distribuir, depurar y administrar aplicaciones PHP críticas de negocios. Permite conectarse directamente con la bases de datos profesionales más utilizadas tales como IBM DB2/Cloudscape/ Derby/, MySQL, Oracle, Microsoft SQL Server, PostgreSQL y SQlite. Tiene características de depuración avanzadas, incluyendo: condiciones límites, visualización de errores, vistas avanzadas, variables y buffer de salida. Asegura la protección máxima de ubicaciones de proyectos o en Internet con depuradores remotos. Facilita el desarrollo y colaboración en equipo mediante la administración efectiva de su código fuente utilizando CVS o Subversión directamente desde Zend Studio. Tiene soporte para PHP 5 completo, analizador de código, carpeta de código, completado de código, coloreado de sintaxis, administrador de proyecto, editor de código, depurador de gráficos y asistentes. [20]

- ✓ **EMS PostgreSQL Manager**

Es una herramienta gráfica de gran alcance para la administración y el desarrollo del servidor de BD PostgreSQL. Permite no sólo ejecutar las consultas y scripts SQL, sino importar y exportar datos de otros formatos, manejar usuarios y privilegios, crear y editar bases de datos y tablas.

1.5.11 Control de versiones

Una versión, revisión o edición de un producto, es el estado en el se encuentra en un momento dado en su desarrollo o modificación. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

Un sistema de control de versiones debe proporcionar un mecanismo de almacenaje de los elementos que deba gestionar y un registro histórico de las acciones realizadas

con cada elemento o conjunto de elementos (normalmente brindando la posibilidad de volver o extraer un estado anterior del producto) entre otros aspectos. Todos los sistemas de control de versiones se basan en disponer de un repositorio, que es el conjunto de información gestionada por el sistema. Este repositorio contiene el historial de versiones de todos los elementos gestionados. Cada uno de los usuarios puede crearse una copia local duplicando el contenido del repositorio para permitir su uso. Es posible duplicar la última versión o cualquier versión almacenada en el historial.

- ✓ **Subversion**

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

1.6 Conclusiones del Capítulo 1

Al finalizar este primer capítulo del presente trabajo se puede concluir que se han tratado los temas necesarios en aras de aportar los conocimientos requeridos sobre los distintos Sistemas de Gestión Empresarial que tienen implementados el módulo Cobros y Pagos, además de las principales actividades que abarcan los procesos de cobros y pagos; así como el modelo de desarrollo utilizado, el lenguaje de programación del lado del servidor PHP, el EXTjs para la implementación del lado del cliente, así como del gestor de bases de datos PostgreSQL, el navegador Mozilla Firefox y el Zend Studio, que por sus características y prestaciones han sido los lenguajes o herramientas definidas por los arquitectos del sistema.

CAPÍTULO 2: DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA.

2.1 Introducción

En este capítulo, se explica el diseño alcanzado como propuesta para desarrollar el sistema debidamente, los principales aspectos que se han tenido en cuenta para la implementación del mismo, teniendo presente la propuesta de los analistas y diseñadores del sistema para lograr así un producto con la calidad y eficiencia requeridas.

El objetivo principal ha sido detallar los principales procesos con los que se deben trabajar para facilitar la comprensión del funcionamiento de los componentes implementados y lograr que el sistema satisfaga las necesidades del usuario final.

2.2 Valoración crítica del diseño propuesto por el analista

Del diseño propuesto por los analistas se pudo extraer las clases fundamentales que deben ser definidas para que el sistema funcione satisfactoriamente, así como los atributos y métodos que deben tener las mismas, dando una idea clara de lo que se debe implementar. Unido a esto se encuentran los diagramas de interacción, que explican la colaboración que existe entre las clases y cómo son llamados los métodos y sentencias dentro de cada una, de los cuales se obtuvo la información necesaria para conocer el orden de las acciones a implementar. Además permitió una mejor comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y tecnologías de interfaz de usuario.

El diseño propuesto fue creado siguiendo patrones, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, permitiendo llevar a cabo la implementación clara y limpia del subsistema bajo patrones como :

- El Proxy que es un patrón de tipo estructural asociado a objetos, que se encuentra entre la interfaz y la implementación e intercepta las llamadas a los métodos. La intención del Proxy es controlar el acceso al objeto deseado, además de mejorar la funcionalidad del mismo; proporciona un sustituto o representante de otro objeto para controlar el acceso a éste. Un objeto Proxy, recibe llamadas a métodos que pertenecen a otros objetos y comparte interfaz o superclase común con el objeto que realmente provee el servicio [21].
- El Decorator que se debe usar para adicionar responsabilidades a objetos individuales dinámicamente sin afectar otros objetos; para agregar

responsabilidades que pueden ser retiradas; o cuando no es práctico adicionar responsabilidades por medio de la herencia.

- El MVC (Modelo Vista Controlador) cuyo objetivo principal es realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad, permitiendo esto que las modificaciones realizadas en las vistas influyan en menor medida en la lógica de negocio o de datos.
- El DAO (Data Access Object) que surge de la necesidad de gestionar las fuentes de datos, aunque su uso se extiende al problema de encapsular no sólo la fuente de datos, sino además ocultar la forma de acceder a los datos. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cual es la fuente de almacenamiento.

2.3 Análisis de posibles implementaciones, componentes o módulos ya existentes.

2.3.1 Frameworks

2.3.1.1 ExtJs Framework

Es un framework JavaScript del lado del cliente para el desarrollo de aplicaciones Web. Tiene un sistema dual de licencia: Comercial y Código Abierto. Este framework puede correr en cualquier plataforma que pueda procesar POST y devolver datos estructurados (PHP, Java, .NET y algunas otras) en tiempo de ejecución carga y crea todos los objetos HTML a través del uso intenso del DOM. Los datos son obtenidos mediante mensajes AJAX a través de XML y/o JSON.

Dispone de un conjunto de componentes para incluir dentro de una aplicación web, como:

- ✓ Cuadros y áreas de texto.
- ✓ Campos para fechas.
- ✓ Campos numéricos.
- ✓ Combobox.
- ✓ Radiobuttons y checkboxs.
- ✓ Editor HTML.
- ✓ Elementos de datos (con modos de sólo lectura, datos ordenables, columnas que se pueden bloquear y arrastrar, etc.).
- ✓ Árbol de datos.
- ✓ Pestañas.
- ✓ Barra de herramientas.
- ✓ Menús al estilo de Windows.
- ✓ Paneles divisibles en secciones.

- ✓ Sliders.

También contiene numerosas funcionalidades que permiten añadir interactividad a las páginas HTML, como:

- ✓ Cuadros de diálogo.
- ✓ Quicktips para mostrar mensajes de validación e información sobre campos individuales.

2.3.1.2 Doctrine Framework

Es un potente y completo sistema ORM (object relational mapper) para PHP 5.2+ que incorpora una DBL (capa de abstracción a base de datos). Uno de sus rasgos importantes es la habilidad de escribir opcionalmente las preguntas de la base de datos orientado a objeto. Esto les proporciona una alternativa poderosa a diseñadores de SQL que mantiene un máximo de flexibilidad sin requerir la duplicación del código innecesario. También permite exportar una base de datos existente a sus clases correspondientes y también convierte clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos.

Ver *Anexo 3*

2.3.1.3 Zend Framework

Es un framework para desarrollo de aplicaciones Web y servicios Web con PHP. Brinda soluciones para construir sitios web modernos, robustos y seguros. Además es Open Source y trabaja con PHP 5.

Presenta entre otras las siguientes características:

- ✓ Proporciona un sistema de caché dividido en frontend y backend, de forma que se puedan almacenar en caché diferentes datos como resultados de funciones, páginas completas, etc., y que esta información se almacene en archivos, en memoria, en base de datos, etc.
- ➢ Simplifica la gestión de archivos de configuración.
- ➢ Proporciona los componentes que forman la infraestructura del patrón MVC.
- ➢ Proporciona una capa de acceso a base de datos, construida sobre PDO pero ampliándola con diferentes características.
- ➢ Proporciona mecanismos de filtrado y validación de entradas de datos.
- ➢ Permite convertir estructuras de datos PHP a JSON y viceversa, para su utilización en aplicaciones AJAX.
- ➢ Proporciona las características necesarias para proveer y consumir servicios web vía REST.
- ➢ Proporciona capacidades de búsqueda sobre documentos y contenidos.
- ➢ Permite consumir y proveer servicios web.

Ver *Anexo 4*

2.3.1.3.1 ZendExt Framework

Es un framework de código abierto, que está diseñado para PHP 5 y tiene buenas capacidades de ampliación. Es elaborado a partir de Zend Framework cumpliendo con todas sus características. Este trae de novedoso un controlador vertical para el control de las acciones realizadas por las vistas hacia el controlador, un motor de reglas para las validaciones en el servidor, se le incluyó el IoC para la comunicación en entre los módulos o componentes. Se le incorporó la integración con el ORM Doctrine Framework para trabajo en la capa de abstracción a base de datos y el ExtJs Framework para el desarrollo de las vistas.

2.3.1.4 UCID Framework

Es el Framework encargado del trabajo con las vistas. Abarca la integración de ExtJs Framework con el sistema incluyendo el integrador de interfaz, el generador de interfaz dinámica y la impresión de documentos. Integra la iconografía, los diferentes temas de escritorio de la aplicación, el multilenguaje.

2.3.2 Arquitectura

La Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema. Una Arquitectura de Software, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

2.3.2.1 Estilo Basado en Capas

El modelo “n−capas” de informática distribuida ha emergido como la arquitectura predominante para la construcción de aplicaciones multiplataforma en la mayor parte de las empresas. Como tecnología, las arquitecturas de n−capas proporcionan una gran cantidad de beneficios para las empresas que necesitan soluciones flexibles y fiables para resolver complejos problemas inmersos en cambios constantes.

Ventajas del modelo:

- ✓ Desarrollos paralelos (en cada capa).
- ✓ Aplicaciones más robustas debido al encapsulamiento.
- ✓ Mantenimiento y soporte más sencillo (es más sencillo cambiar un componente que modificar una aplicación monolítica).

- ✓ Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- ✓ Alta escalabilidad. La principal ventaja de una aplicación distribuida bien diseñada es su buen escalado, es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

2.3.2.2 Modelo-Vista –Controlador

MVC es un patrón de diseño de arquitectura de software usado principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de los conceptos para que el desarrollo este estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente. MVC sugiere la separación del software en 3 estratos.

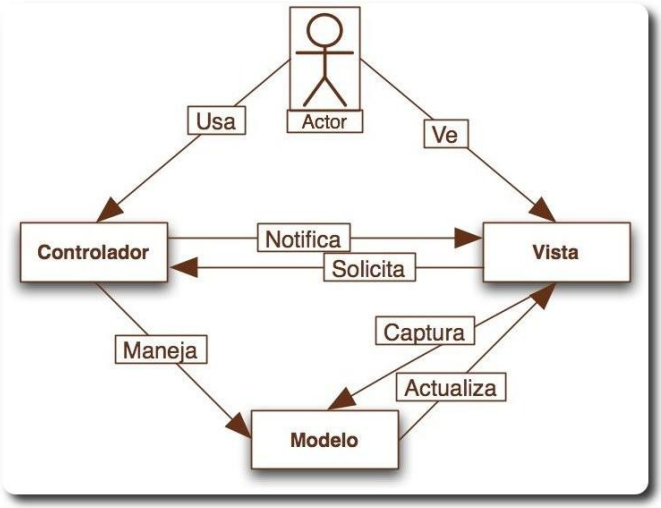


Figura 2 Estructura Modelo-Vista-Controlador

Modelo: Es la representación de la información que maneja la aplicación. El modelo en si son los datos puros que puestos en un contexto del sistema proveen de información al usuario o a la aplicación misma.

Vista: Es la representación del modelo en forma gráfica, disponible para la interacción con el usuario. En el caso de una aplicación web la "Vista " es la página con contenido dinámico sobre el cual el usuario puede realizar operaciones.

Controlador: Es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el Modelo en caso de ser necesario.

2.3.2.3 Arquitectura Cliente - Servidor

Cuando se habla de aplicaciones Web es preciso pensar en la arquitectura más eficaz que permita el control e intercambio de información a través de la red. La Arquitectura Cliente-Servidor es una de las más importantes y usadas en este ámbito de enviar y recibir información, como su nombre lo indica está compuesta por un cliente y un servidor, donde el cliente realiza una petición de recursos, información o servicios al servidor, y este último se encarga de proporcionar al cliente la respuesta a las peticiones realizadas. Una de las ventajas de esta arquitectura es que el acceso a la información se realiza de forma más ágil y al estar almacenada en el servidor existe un mejor control de la seguridad. [10]

Ver *Anexo 2*

Los clientes realizan generalmente funciones como:

- Manejo de la interfaz de usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.

Por su parte los servidores realizan, entre otras, las siguientes funciones:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local.
- Siempre que un cliente requiere un servicio lo solicita al servidor correspondiente y éste le responde proporcionándolo. Normalmente, pero no necesariamente, el cliente y el servidor están ubicados en distintos procesadores. Los clientes se suelen situar en ordenadores personales y/o estaciones de trabajo y los servidores en procesadores departamentales o de grupo.

Entre las principales características de la arquitectura cliente - servidor se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

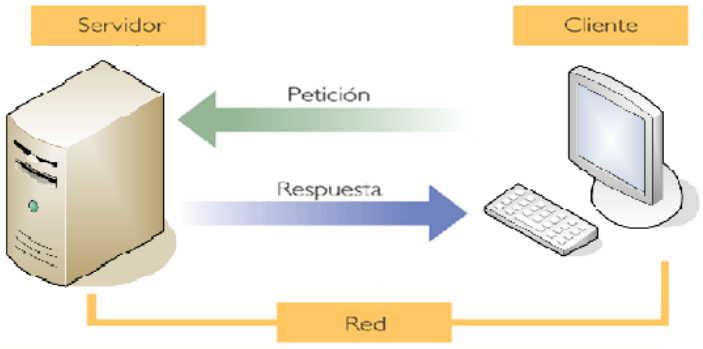


Figura 3 Representación de la Arquitectura Cliente - Servidor.

2.3.3 Estrategias de integración

La aplicación esta definida por tres capas: capa de Presentación (view), Negocio (controller) y Acceso a Datos (models), esta arquitectura posibilita un trabajo seguro, rápido y eficiente. La integración vertical o llamada arquitectura en 3 capas consiste en el flujo de los datos desde la vista hacia la capa de datos y viceversa, pasando por los diferentes elementos que componen la arquitectura. Esta consta de cuatro nodos de integración, el que encontramos entre la vista y el controlador, el que está entre el controlador y el modelo, el que vincula el modelo con el framework doctrine y el que se encuentra entre el doctrine y la base de datos. Todo el código dentro un mismo componente utiliza llamadas a métodos o eventos de forma directa. La comunicación entre diferentes módulos y componentes se realiza mediantes llamadas a la inversión de control. El IoC especifica respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que otro módulo o componente lleve a cabo las acciones de control que se requieran en el orden necesario para el conjunto de sucesos que tengan que ocurrir.

Cada componente tiene su registro de los datos de los módulos en un fichero xml que será mapeado por el framework para el funcionamiento del mismo, dicho fichero tiene por nombre IoC y registra las funcionalidades que ofrecen los métodos de las clases controladoras de los componentes del sistema. La base de datos es accedida de forma directa mediante controladoras y los componentes rehusados son integrados mediantes interfaces sencillas, garantizando así una total integración de las capas en el sistema.

Ver *Anexo 5*

2.4 Estándares de código.

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Los estándares de codificación

permiten una mejor integración entre las líneas de producción y establece pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

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

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

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

- ✓ Las cláusulas, es decir, la notación que se utilizará para nombrar cada uno de los identificadores que se declaran.
- ✓ La estructura del código en sí, es decir, lo referente a las tabulaciones y los espacios entre líneas, y dentro de las líneas, los espacios entre los operadores y estructuras que componen el lenguaje en que programamos.

Los estándares de codificación utilizados fueron:

2.4.1 Notación húngara

Esta convención se basa en definir prefijos para cada tipo de datos y según el ámbito de las variables. También es conocida como notación: REDDICK (por el nombre de su creador). La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definiendo en ella un prefijo que identifique su tipo de dato y ámbito. Esta notación se utilizó para los nombres de las variables. A continuación un ejemplo:

intEdad: Según la definición vemos que esta variable es de tipo INTEGER y que representa la edad de alguna persona.

Los prefijos a utilizar en la creación de variables serán los siguientes

Tipos de Datos	Prefijos
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
Float	flt
Boolean	boo

Tabla 1: Prefijos para la creación de variables.

2.4.2 Notación PascalCasing

Es como la notación húngara pero sin prefijos. En este caso, los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. Esta notación se utilizó para los nombres de las clases. A continuación un ejemplo:

GestionarUsuario: Este nombre de clase esta compuesto por 2 palabras, ambas iniciando con letra mayúscula.

2.4.3 Notación CamelCasing

Es parecido al Pascal-Casing con la excepción que la letra inicial del identificador no debe estar en mayúscula. Esta notación se utilizó para el nombre de las funciones y el nombre de los atributos. A continuación ejemplos:

insertarMoneda: Este nombre de método esta compuesto por 2 palabras, la primera todo en minúsculas y la segunda iniciando con letra mayúscula.

dineroM: Este nombre del atributo esta compuesto por 2 palabras, la primera todo en minúsculas y la segunda iniciando con letra mayúscula.

2.5 Descripción de los algoritmos no triviales a implementar.

Uno de los componentes del módulo de Cobros y Pagos es conciliación, las cuales pueden ser de derechos u obligaciones, o de anticipos. El método adicionarconciliacionAction() pertenece a la conciliación de derecho u obligación, el cual se describe a continuación:

1. Se recibe un arreglo de objetos con todos los derechos u obligaciones a conciliar, lo cual incluye si se le cambia el plazo o se cancela. Además se declaran para posterior uso un arreglo *arrayiderecho*, otro *arrayiddoc* y una variable *iddocumento* inicializada en cero.
2. Para cada posición del arreglo:
   2.1 Se crea el objeto *DatConciliacion* y se salva en la base de datos (BD), además se guarda una referencia de dicha operación en la BD.
   2.2 Si los atributos *coincide* y *cambio de plazo* del objeto tienen los valores “Si” y “Cambio plazo” respectivamente, se obtienen mediante IoC la condición de pago correspondiente al cual se le modifica el campo “plazo” y se inserta nuevamente en la BD.
   2.3 Si los atributos *coincide* y *cambio de plazo* del objeto tienen los valores “No” y “Cancelar” respectivamente, se salva el arreglo *arrayiderecho* el indetificador de *derechouobligacion*. Se comprueba que la variable *iddocumento* se mantenga en cero, se procede a realizar el cambio de cuenta y a insertar un nuevo documento en la BD salvando su identificador en la variable *iddocumento*.
3. Si la variable iddocumento tiene valor diferente de cero se guarda su valor en *arrayiddoc*.
4. Se obtienen la cantidad de operaciones referentes el cambio de cuenta. Si las mismas son mayores que cero se recorre el arreglo de operaciones y se salva en la variable *tipooperbyalias* el identificador que corresponde a la operación “litigio”.
5. Se recorre el arreglo *arrayiderecho*, insertando el cambio de cuenta y cambiando el estado a cancelado para cada posición.
6. Si la cantidad de *iddocumento* es mayor que cero se procede a contabilizar el cambio de cuenta.
7. Se muestra el mensaje: “La conciliación fue registrada satisfactoriamente.”

2.5.1 Análisis de complejidad de los mismos

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

método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y proporcionando el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. Es una de las métricas de software mas ampliamente aceptada, ya que ha sido concebida para ser independiente del lenguaje. Se ha medido un gran número de programas, de modo de establecer rangos de complejidad que ayuden al ingeniero de software a determinar la estabilidad y riesgo inherente de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad. Algunos estudios experimentales indican la existencia de distintas relaciones entre la métrica de McCabe y el número de errores existentes en el código fuente, así como el tiempo requerido para encontrar y corregir esos errores. Se suele comparar la complejidad ciclomática obtenida contra un conjunto de valores límite como se observa en la **tabla 2**. [23]

Complejidad Ciclomática	Evaluación de Riesgo
1-10	Programa simple, sin mucho riesgo
11-20	Más Complejo, riesgo moderado
21-50	Complejo, programa de alto riesgo
50	Programa no testeable, muy alto riesgo

Tabla 2: Complejidad ciclomática Vs evaluación de riesgo

Para conocer la complejidad del algoritmo es necesario calcular la complejidad ciclomática del mismo, para hacer dicho cálculo es necesario primero tener el código o el diseño del algoritmo, luego enmarcar cada instrucción del código con un número, que representa cada lugar del camino que puede seguir la secuencia del algoritmo.
Ver Anexo 6

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

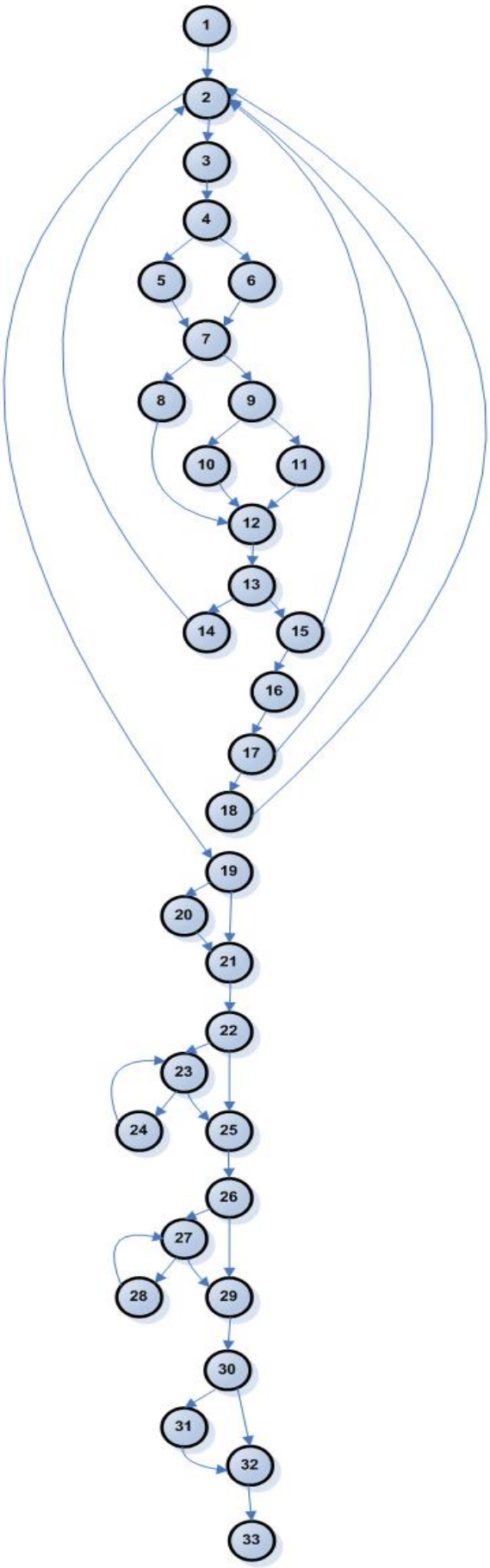


Figura 4: Grafo de flujo del algoritmo adicionarconciliacionAction().

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

- ✓ V (G) = A – N + 2

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

V (G) = (45 – 33) + 2

V (G) = 14

- ✓ V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 13 +1

V (G) = 14

- ✓ V (G) = R

Siendo “R” la cantidad total de regiones, para cada formula “V (G)” representa el valor del calculo.

V (G) = 14

Realizado el cálculo por las tres vías necesarias se llega a la conclusión que el algoritmo presentado anteriormente tiene una complejidad ciclomática de 14, dando una visión de que existen a lo sumo catorce caminos lógicos por donde recorrer el algoritmo. Al tener una complejidad catorce, la evaluación de riesgo nos dice que es un algoritmo más complejo, con riesgo moderado.

2.6 Estructuras de datos apropiadas para la implementación de estos algoritmos.

En programación, una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema. Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De esta forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

En la realización de un script en PHP en múltiples ocasiones existen variables que tienen información similar y se procesan de forma semejante. Para ello PHP posee un elemento denominado array. Un array es un conjunto de variables agrupadas bajo un único nombre. Cada variable dentro de la matriz se denomina elemento. Dentro de la misma matriz pueden existir variables de diferentes tipos

Una matriz en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia valores con claves. Este tipo es optimizado en varias formas, de modo que puede usarlo como una matriz real, o una lista (vector), tabla asociativa (caso particular de implementación de un mapa), diccionario, colección, pila, cola y probablemente más. Ya que puede tener otra matriz PHP como valor, es realmente fácil simular árboles.

2.7 Descripción de las nuevas clases u operaciones necesarias.

2.7.1 Clases del componente: Submayor

Nombre: SubmayorfiscalController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
submayorfiscalAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar submayor fiscal y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar todas las monedas de la entidad.
cargarcuentaAction ()	Responsabilidad encargada de cargar todas las cuentas de la entidad.

Nombre: SubmayorclienteprovedorController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	

Nombre.	Descripción.
init()	Constructor de la clase.
submayorclienteprovvedorAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar submayor cliente y proveedor y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar todas las monedas de la entidad.
cargarcuentaAction ()	Responsabilidad encargada de cargar todas las cuentas de la entidad.
cargarclienteAction()	Responsabilidad encargada de cargar los clientes de la entidad.
cargarsubmyAction()	Responsabilidad encargada de cargar los submayores.

Nombre: SubmayorclienteModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
SubmayorclienteModel()	Constructor de la clase.
MostrarSumayor()	Responsabilidad encargada de cargar los submayores dados la cuenta, el cliente, el idmoneda y el idmonedacont.
comparaFechas()	Responsabilidad encargada de comparar dos fechas y devolver la diferencia entre ellas.

2.7.2 Clases del componente: Derecho u obligación

Nombre: GestionarobligacionesfiscalesController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**

init()	Constructor de la clase.
gestionarobligacionesfiscalesAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar obligaciones fiscales y redireccionar a la interfaz.
adicionarobligacionfiscalAction ()	Responsabilidad encargada de adicionar una obligación fiscal
modificarobligacionfiscalAction ()	Responsabilidad encargada de modificar una obligación fiscal con los nuevos valores introducidos por el usuario.
confirmarobligacionfiscalAction ()	Responsabilidad encargada de confirmar las obligaciones fiscales seleccionadas por el usuario.
contabilizarobligacionfiscalAction ()	Responsabilidad encargada de contabilizar las obligaciones fiscales seleccionadas por el usuario.
cargaroperacionesbydcopAction ()	Responsabilidad encargada de cargar todas las operaciones que tiene asociada una determinada obligación fiscal.
cargarobligacionesfiscalesAction ()	Responsabilidad encargada de cargar todas las obligaciones fiscales.
cargarcondicionpagobyidAction ()	Responsabilidad encargada de cargar la condición de pago asociada a una determinada obligación fiscal.
cargarregistroanexoalpaseAction()	Responsabilidad encargada de cargar los anexos asociados a una operación.
cargaroperacionesiocAction ()	Responsabilidad encargada de cargar todas las operaciones de manera que el usuario las pueda seleccionar.
cargarmonedaiocAction()	Responsabilidad encargada de cargar todas las monedas de manera que el usuario las pueda seleccionar.
cargartasayfactordeconversioniocAction ()	Responsabilidad encargada cargar la tasa y el factor de conversión de una determinada moneda.
cargarcuentasiocAction ()	Responsabilidad encargada de cargar todas las cuentas.
cargartipodocumentoiocAction ()	Responsabilidad encargada de cargar los tipos de documentos existentes de manera que el

	usuario los pueda seleccionar.
cargarparrafofiscaliocAction ()	Responsabilidad encargada de cargar los párrafos fiscales existentes de manera que el usuario los pueda seleccionar.
cargarcondiciondepagoiocAction ()	Responsabilidad encargada de cargar las condiciones de pago establecidas por defecto.
centrocostoAction()	Responsabilidad encargada de cargar los centros de costo de manera que el usuario los pueda seleccionar.
objetogastoAction()	Responsabilidad encargada de cargar los objetos de gasto de manera que el usuario los pueda seleccionar.
cargarimportecontableAction()	Responsabilidad encargada de calcular el importe contable.
importembanexosAction()	Responsabilidad encargada de calcular el importe contable de todos los anexos.
cargarobligacionpagofiscalAction()	Responsabilidad encargada de cargar las obligaciones fiscales que se pueden cancelar.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivo de cancelación.
cancelaobligacionpagofiscalAction()	Responsabilidad encargada de cancelar las obligaciones fiscales seleccionadas por el usuario.

Nombre: GestionarobligacionpagoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestionarobligacionpagoAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar obligaciones de pago y redireccionar a la interfaz.
cargarcondpagoAction()	Responsabilidad encargada de cargar la condición de pago establecida por defecto.

cargarcondpagoidAction()	Responsabilidad encargada de cargar una condición de pago dado su id.
adiobligaciondepagoAction ()	Responsabilidad encargada de adicionar una obligación de pago.
formatonumeroAction()	Responsabilidad encargada de darle el formato correcto a los importes.
AuxIdCondPago()	Responsabilidad encargada de encontrar cual es el id de una condición de pago.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de cuentas.
modobligaciondepagoAction ()	Responsabilidad encargada de modificar una obligación de pago con los nuevos datos introducidos por el usuario.
cargaroperacionesAction()	Responsabilidad encargada de cargar todas las operaciones asociadas a una obligación de pago.
condicionpagoAction()	Responsabilidad encargada de cargar la condición de pago establecida por defecto.
cargarmonedaAction()	Responsabilidad encargada de cargar todas las monedas de manera que el usuario las pueda seleccionar.
cargartasaAction()	Responsabilidad encargada de cargar todas las tasas asociadas a una moneda de manera que el usuario las pueda seleccionar.
confirmarAction()	Responsabilidad encargada de confirmar todas las obligaciones de pago seleccionadas por el usuario.
cargaroperacionAction()	Responsabilidad encargada de cargar todas las operaciones de manera que el usuario las pueda seleccionar.
cargarclienteAction()	Responsabilidad encargada de cargar todos los clientes de manera que el usuario los pueda seleccionar.
cargardocumentoAction()	Responsabilidad encargada de cargar todos los documentos de manera que el usuario los pueda seleccionar.
cuentaaperturaporclienteAction()	Responsabilidad encargada de comprobar si una cuenta tiene apertura por un determinado cliente.

Cargardocumentobyid()	Responsabilidad encargada de cargar un tipo de documento conocido su id.
cargarderuobliAction()	Responsabilidad encargada de cargar todos los derechos de cobro.
cuentasporoperacionAction()	Responsabilidad encargada de encontrar la cuenta asociada a una operación.
cuentaporidAction()	Responsabilidad encargada de verificar si una cuenta es da gasto o no.
centrocostoAction()	Responsabilidad encargada de cargar los centros de costo.
objetogastoAction()	Responsabilidad encargada de cargar los objetos de gasto.
cargaranexosAction()	Responsabilidad encargada de cargar los registros de anexo asociado a una operación.
cargarderechocobroclienteAction()	Responsabilidad encargada de cargar las obligaciones de pago que se pueden cancelar.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivo de cancelación.
canceladerechocobroclienteAction()	Responsabilidad encargada de cancelar las las obligaciones de pago seleccionados por el usuario.
contabilizarAction()	Responsabilidad encargada de contabilizar las obligaciones de pago seleccionadas por el cliente.

Nombre: GestionarderechofiscalController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestionarderechofiscalAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar derechos

	fiscales y redireccionar a la interfaz.
adicionarderechofiscalAction ()	Responsabilidad encargada de adicionar un derecho fiscal.
modificarderechofiscalAction ()	Responsabilidad encargada de modificar un derecho fiscal con los nuevos valores introducidos por el usuario.
confirmarderechofiscalAction ()	Responsabilidad encargada de confirmar los derechos fiscales seleccionados por el usuario.
contabilizarderechofiscalAction ()	Responsabilidad encargada de contabilizar los derechos fiscales seleccionados por el usuario.
cargaroperacionesbydcopAction ()	Responsabilidad encargada de cargar todas las operaciones que tiene asociada un determinado derecho fiscal.
cargarderechosfiscalesAction ()	Responsabilidad encargada de cargar todos los derechos fiscales.
cargarcondicionpagobyidAction ()	Responsabilidad encargada de cargar la condición de pago asociada a un determinado derecho fiscal.
cargarregistroanexoalpaseAction()	Responsabilidad encargada de cargar los anexos asociados a una operación.
cargaroperacionesiocAction ()	Responsabilidad encargada de cargar todas las operaciones de manera que el usuario las pueda seleccionar.
cargarmonedaiocAction()	Responsabilidad encargada de cargar todas las monedas de manera que el usuario las pueda seleccionar.
cargartasayfactordeconversioniocAction ()	Responsabilidad encargada cargar la tasa y el factor de conversión de una determinada moneda.
cargarcuentasiocAction ()	Responsabilidad encargada de cargar todas las cuentas.
cargartipodocumentoiocAction ()	Responsabilidad encargada de cargar los tipos de documentos existentes de manera que el usuario los pueda seleccionar.
cargarparrafofiscaliocAction ()	Responsabilidad encargada de cargar los párrafos fiscales existentes de manera que el usuario los pueda seleccionar.
cargarcondiciondepagoiocAction ()	Responsabilidad encargada de cargar las

	condiciones de pago establecidas por defecto.
centrocostoAction()	Responsabilidad encargada de cargar los centros de costo de manera que el usuario los pueda seleccionar.
objetogastoAction()	Responsabilidad encargada de cargar los objetos de gasto de manera que el usuario los pueda seleccionar.
cargarimportecontableAction()	Responsabilidad encargada de calcular el importe contable.
importembanexosAction()	Responsabilidad encargada de calcular el importe contable de todos los anexos.
cargarobligacionpagofiscalAction()	Responsabilidad encargada de cargar los derechos fiscales que se pueden cancelar.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivo de cancelación.
cancelaobligacionpagofiscalAction()	Responsabilidad encargada de cancelar las obligaciones fiscales seleccionadas por el usuario.

Nombre: GestionarderechocobroController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestionarderechocobroAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar derechos de cobro y redireccionar a la interfaz.
cargarcondpagoAction()	Responsabilidad encargada de cargar la condición de pago establecida por defecto.
cargarcondpagoidAction()	Responsabilidad encargada de cargar una condición de pago dado su id.
adiderechocobroAction()	Responsabilidad encargada de adicionar un derecho de cobro.
formatonumeroAction()	Responsabilidad encargada de darle el formato

	correcto a los importes.
AuxIdCondPago()	Responsabilidad encargada de encontrar cual es el id de una condición de pago.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de cuentas.
modderechocobroAction()	Responsabilidad encargada de modificar un derecho de cobro con los nuevos datos introducidos por el usuario.
cargaroperacionesAction()	Responsabilidad encargada de cargar todas las operaciones asociadas a un derecho de cobro.
condicionpagoAction()	Responsabilidad encargada de cargar la condición de pago establecida por defecto.
cargarmonedaAction()	Responsabilidad encargada de cargar todas las monedas de manera que el usuario las pueda seleccionar.
cargartasaAction()	Responsabilidad encargada de cargar todas las tasas asociadas a una moneda de manera que el usuario las pueda seleccionar.
confirmarAction()	Responsabilidad encargada de confirmar todos los derechos de cobro seleccionados por el usuario.
cargaroperacionAction()	Responsabilidad encargada de cargar todas las operaciones de manera que el usuario las pueda seleccionar.
cargarclienteAction()	Responsabilidad encargada de cargar todos los clientes de manera que el usuario los pueda seleccionar.
cargardocumentoAction()	Responsabilidad encargada de cargar todos los documentos de manera que el usuario los pueda seleccionar.
cuentaaperturaporclienteAction()	Responsabilidad encargada de comprobar si una cuenta tiene apertura por un determinado cliente.
Cargardocumentobyid()	Responsabilidad encargada de cargar un tipo de documento conocido su id.
cargarderuobliAction()	Responsabilidad encargada de cargar todos los derechos de cobro.
cuentasporoperacionAction()	Responsabilidad encargada de encontrar la cuenta asociada a una operación.

cuentaporidAction()	Responsabilidad encargada de verificar si una cuenta es da gasto o no.
centrocostoAction()	Responsabilidad encargada de cargar los centros de costo.
objetogastoAction()	Responsabilidad encargada de cargar los objetos de gasto.
cargaranexosAction()	Responsabilidad encargada de cargar los registros de anexo asociado a una operación.
cargarderechocobroclienteAction()	Responsabilidad encargada de cargar los derechos de cobro que se pueden cancelar.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivo de cancelación.
canceladerechocobroclienteAction()	Responsabilidad encargada de cancelar las los derechos de cobros seleccionados por el usuario.
contabilizarAction()	Responsabilidad encargada de contabilizar los derechos de cobro seleccionados por el cliente.

Nombre: DatClienteModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
DatClienteModel ()	Constructor de la clase.
EliminaCliente ()	Responsabilidad encargada de eliminar un cliente dado el objeto.

Nombre: DatOperacioncpModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	

Nombre.	Descripción.
DatOperacioncpModel ()	Constructor de la clase.
EliminaOperacion ()	Responsabilidad encargada de eliminar una operación dado el objeto.
InsertarOperacion()	Responsabilidad encargada de insertar una operación dado el objeto.
ModificarOperacion()	Responsabilidad encargada de modificar una operación dado el objeto.
EliminarOperacionPorId()	Responsabilidad encargada de eliminar una operación dado su Id.

Nombre: derechosdecobroModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
derechosdecobroModel ()	Constructor de la clase.
insertarderechodecobro ()	Responsabilidad encargada de insertar un derecho de cobro dado el objeto.
Modificarderechodecobro()	Responsabilidad encargada de modificar un derecho de cobro dado el objeto.

Nombre: obligacionpagoModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
obligacionpagoModel ()	Constructor de la clase.
insertarob ()	Responsabilidad encargada de insertar una obligación de pago dado el objeto.
modificarob ()	Responsabilidad encargada de modificar una obligación de pago dado el objeto.

Nombre: ContabilizarDerechoObligacion.

Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
ContabilizarDerechoObligacion ()	Constructor de la clase.
Contabilizar ()	Responsabilidad encargada de contabilizar y derecho u obligación.

Nombre: DatDerechouobligacionModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
DatDerechouobligacionModel ()	Constructor de la clase.
BuscarDerechouobligacionById ()	Responsabilidad encargada de buscar derechos u obligación dado el id.
Actualizar ()	Responsabilidad encargada de actualizar un derecho u obligación dado el objeto.
ObtenerDerechoUObligacionPorId()	Responsabilidad encargada de buscar derechos u obligación dado el id y el cliente
InsertarDerechoUObligacion()	Responsabilidad encargada de insertar derechos u obligaciones dado un arreglo de objetos.
ModificarDerechoUObligacion()	Responsabilidad encargada de modificar derechos u obligaciones dado un arreglo de objetos.
EliminarDerechoUObligacion()	Responsabilidad encargada de eliminar un derecho u obligación dado el id.

Nombre: derechosfiscalesModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**

Para cada responsabilidad	
Nombre.	**Descripción.**
derechosfiscalesModel ()	Constructor de la clase.
adicionarderechofiscal ()	Responsabilidad encargada de adicionar derechos fiscales.
modificarderechofiscal ()	Responsabilidad encargada de modificar derechos fiscales.
confirmarderechofiscal ()	Responsabilidad encargada de confirmar derechos fiscales.
contabilizarderechofiscal ()	Responsabilidad encargada de contabilizar derechos fiscales.
cargarderechosfiscales ()	Responsabilidad encargada de cargar derechos fiscales.
Cargarcondicionpagoderecho()	Responsabilidad encargada de cargar condiciones de pago dado el id.
Cargaroperacionesdcopmodel()	Responsabilidad encargada de cargar las operaciones de un derecho dado el id.
Cargaranexosbyidoperacion()	Responsabilidad encargada de cargar registros de Anexo al Pase.

Nombre: obligacionesfiscalesModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
obligacionesfiscalesModel ()	Constructor de la clase.
adicionarobligfiscal ()	Responsabilidad encargada de insertar una obligación fiscal.
modificarobligfiscal ()	Responsabilidad encargada de modificar una obligación fiscal.
confirmarobligfiscal ()	Responsabilidad encargada de confirmar una obligación fiscal.
contabilizarobligacionfiscal ()	Responsabilidad encargada de contabilizar obligaciones fiscales dados sus Id.
cargarobligacionesfiscales ()	Responsabilidad encargada de cargar las obligaciones fiscales.

Cargarcondicionpagoobligacion()	Responsabilidad encargada de cargar una condición de pago dado su id.
Cargaroperacionesdcopmodel()	Responsabilidad encargada de cargar operaciones de una obligación dado su id.
Cargaranexosbyidoperacion()	Responsabilidad encargada de cargar registros de Anexo al Pase.

2.7.3 Clases del componente: Funcionalidades

Nombre: CambiarcuentaoefectoalitigioController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
cambiarcuentaoefectoalitigioAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar cambios de cuentas y redireccionar a la interfaz.
cargarcombodocAction ()	Responsabilidad encargada de cargar los tipos de documentos, si son de derecho de cobro u obligación de pago.
cargarcomboopAction ()	Responsabilidad encargada de cargar las operaciones configuradas para cambiar una cuenta.
cargarclienteAction ()	Responsabilidad encargada cargar el nomenclador de clientes.
buscarDocAction ()	Responsabilidad encargada de Busca los documentos ha mostrar en un Grid según los filtros seleccionados en la interfaz.
cambiarCtaAction	Responsabilidad encargada de cambiar las cuenta.

Nombre: GestgestionaroperacionesanticipadasController.
Tipo de clase: Controladora.

Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestgestionaroperacionesanticipadascobroAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar operaciones anticipadas y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos de instrumentos de pago anticipados dado el alias.
cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbClienteAction ()	Responsabilidad encargada de cargar los clientes.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
cargarinstrumentoscobroAction ()	Responsabilidad encargada de cargar las obligaciones de pago anticipado.
cargarpagoanticipadoAction()	Responsabilidad encargada de cargar todos los datos de los instrumentos de pago anticipados existentes en la Base de Datos.
CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de pago seleccionado.
liquidarpagoanticipadoAction()	Responsabilidad encargada de liquidar un pago anticipado.
modificarIntAction()	Responsabilidad encargada de modificar los datos del instrumento de pago entrado al sistema.
cargarcobroanticipadoAction()	Responsabilidad encargada de cargar los datos de los instrumentos de pago que se pueden cancelar con su cliente asociado.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivos de cancelación.

cancelacobroanticipadoAction()	Responsabilidad encargada de cancelar las operaciones de pago anticipadas.
contabilizarAction()	Responsabilidad encargada de crear un comprobante y cambiar el estado del instrumento de la obligación de pago anticipada ha contabilizado.
confirmarAction()	Responsabilidad encargada de confirmar el estado de un instrumento de obligación de pago.
contabilizarDocumento()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de obligación de pago anticipado.

Nombre: GestgestionaroperacionesanticipadascobroController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestgestionaroperacionesanticipadascobroAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar operaciones anticipadas y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos de instrumentos de cobro anticipados dado el alias.
cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbClienteAction ()	Responsabilidad encargada de cargar los clientes.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
cargarinstrumentoscobroAction ()	Responsabilidad encargada de cargar los derechos de cobro anticipado.
cargarpagoanticipadoAction()	Responsabilidad encargada de cargar todos los datos de los instrumentos de cobros anticipados existentes en la Base de Datos.

CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de cobro seleccionado.
liquidarpagoanticipadoAction()	Responsabilidad encargada de liquidar un cobro anticipado.
modificarIntAction()	Responsabilidad encargada de modificar los datos del instrumento de cobro entrado al sistema.
cargarcobroanticipadoAction()	Responsabilidad encargada de cargar los datos de los instrumentos de cobros que se pueden cancelar con su cliente asociado.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargarmotivocancelacionAction()	Responsabilidad encargada de cargar los motivos de cancelación.
cancelacobroanticipadoAction()	Responsabilidad encargada de cancelar los derechos de cobro anticipados.
contabilizarAction()	Responsabilidad encargada de crear un comprobante y cambiar el estado del instrumento del derecho de cobro anticipado ha contabilizado.
confirmarAction()	Responsabilidad encargada de confirmar el estado de los instrumento de derecho de cobro.
contabilizarDocumento()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de derecho de cobro anticipado.

Nombre: GestregistrarinstrumentocobroopercionfiscalController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestregistrarinstrumentocobroopercionfiscalAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar Instrumento de

	cobro de operaciones fiscales y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos de instrumentos de cobro de operaciones fiscales dado el alias.
cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de cobro de operaciones fiscales seleccionado.
liquidarinstrumentocobrooperacionesfiscalesAction ()	Responsabilidad encargada de liquidar un cobro de operaciones fiscales.
GuardarPago ()	Responsabilidad encargada de guardar los datos entrados del instrumento de cobro de operaciones fiscales.
cargarinstrumentocobrooperacionesfiscalesAction ()	Responsabilidad encargada de cargar las operaciones de derecho de Cobro de los instrumentos de operaciones fiscales a liquidar.
contabilizarDocumento()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de cobro de operaciones fiscales.

Nombre: GestregistrarinstrumentooperacionesfiscalesController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestregistrarinstrumentooperacionesfiscalesAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar Instrumento de pago de operaciones fiscales y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos

	de instrumentos de pago de operaciones fiscales dado el alias.
cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de pago de operaciones fiscales seleccionado.
liquidarinstrumentocobrooperacionesfiscalesAction ()	Responsabilidad encargada de liquidar un pago de operaciones fiscales.
GuardarPago ()	Responsabilidad encargada de guardar los datos entrados del instrumento de pago de operaciones fiscales.
cargarinstrumentocobrooperacionesfiscalesAction ()	Responsabilidad encargada de cargar las operaciones de derecho de pago de los instrumentos de operaciones fiscales a liquidar.
contabilizarDocumento ()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de pago de operaciones fiscales.

Nombre: GestregistrainstrumentoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestregistrainstrumentoAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar Instrumento de pago y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos de instrumentos de pago dado el alias.
cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbClienteAction()	Responsabilidad encargada de cargar todos los

	clientes de la entidad.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de pago seleccionado.
guardarAction()	Responsabilidad encargada de mandar a salvar los datos entrados en la interfaz.
GuardarPago()	Responsabilidad encargada de guardar los datos del instrumento de pago anticipado en el sistema.
cargarinstrumentospagoAction()	Responsabilidad encargada de cargar las operaciones de obligación de Pago del instrumente a liquidar.
liquidarpagoanticipadoAction()	Responsabilidad encargada de liquidar la obligación de pago.
contabilizarDocumento()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de obligación pago.
confirmarAction()	Responsabilidad encargada de confirmar el estado de los instrumentos de obligación pago.
contabilizarAction()	Responsabilidad encargada de contabilizar un instrumento de obligación pago.

Nombre: GestregistrainstrumentocobroController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestregistrainstrumentocobroAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar Instrumento de cobro y redireccionar a la interfaz.
cargarcbtipoInstAction ()	Responsabilidad encargada de cargar los tipos de instrumentos de pago dado el alias.

cargarcbMonedaAction ()	Responsabilidad encargada de cargar todas las monedas dada la entidad.
cargarcbClienteAction()	Responsabilidad encargada de cargar todos los clientes de la entidad.
cargarcbCuentaAction ()	Responsabilidad encargada de cargar todas las cuentas dada la entidad.
CalculaImportembAction ()	Responsabilidad encargada de calcular el importe moneda base del instrumento de cobro seleccionado.
guardarAction()	Responsabilidad encargada de mandar a salvar los datos entrados en la interfaz.
GuardarPago()	Responsabilidad encargada de guardar los datos del instrumento de cobro anticipado en el sistema.
cargarinstrumentoscobroAction ()	Responsabilidad encargada de cargar las operaciones de derecho de cobro del instrumente a liquidar.
liquidarpagoanticipadoAction()	Responsabilidad encargada de liquidar el derecho de cobro.
contabilizarDocumento()	Responsabilidad encargada de contabilizar la liquidación de los instrumentos de derecho de cobro.
confirmarAction()	Responsabilidad encargada de confirmar el estado de los instrumentos de derecho de cobro.
contabilizarAction()	Responsabilidad encargada de contabilizar un instrumento de derecho de cobro.

Nombre: GestionarcontratoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
obtenertodoscontratosAction()	Responsabilidad encargada de obtener todos los contratos hechos por la entidad.

confirmarcontratosAction()	Responsabilidad encargada de confirmar un contrato ya guardado.
cargarclienteAction()	Responsabilidad encargada de obtener todos los clientes.
cargarmonedaAction()	Responsabilidad encargada de obtener todos los tipos de monedas que existen para realizar el contrato hacia esa moneda.
cargarderechosuobligacionesAction()	Responsabilidad encargada de obtener todos los derechos u obligaciones existentes.
cargaroperacionAction()	Responsabilidad encargada de obtener todas las operaciones existentes de un derecho.
adicionarContratoAction()	Responsabilidad encargada de adicionar un nuevo contrato para la entidad.
modificarcontratoAction()	Responsabilidad encargada de modificar un contrato ya adicionado previamente.
obtenercontratomodificarAction()	Responsabilidad encargada de obtener los atributos de un contrato para modificarlo.
obteneroperacionmodificarAction()	Responsabilidad encargada de obtener los datos reales de las operaciones ya insertadas para ese derecho en ese contrato.
contabilizarcontratoAction()	Responsabilidad encargada de contabilizar un contrato ya confirmado.

Nombre: ContabilizarCambioCuenta	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
ContabilizarDO ()	Responsabilidad encargada de contabilizar los documentos de cambio de cuenta de los derechos y obligaciones.
ContabilizarCP ()	Responsabilidad encargada de contabilizar los documentos de cambio de cuenta de los anticipos.

Nombre: ContabilizarCambioMoneda	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
Contabilizar ()	Responsabilidad encargada de contabilizar los cambios de moneda y emitir un comprobante.
ConformarPases ()	Responsabilidad encargada de crear los pases.

Nombre: CancelacionModel	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
adicionarcondicion ()	Responsabilidad encargada de `crear una cancelación.

Nombre: ContabilizarCancelacionCobroPagoAnticipado	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
Contabilizar ()	Responsabilidad encargada de contabilizar las cancelaciones de cobro y pago anticipados y emitir un comprobante.

Nombre: ContabilizarCancelacionDerechoObligacion	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
Contabilizar ()	Responsabilidad encargada de contabilizar las

	cancelaciones de derecho y obligación.

Nombre: DerechouObligacionModel	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
__construct ()	Responsabilidad encargada de construir la clase.
adicionarMotivoCancelacion()	Responsabilidad encargada de crear un motivo de cancelación.
eliminarMotivoCancelacion()	Responsabilidad encargada de eliminar un motivo de cancelación.

Nombre: ContabilizarLiquidacionCobroPagoDerechoObligacion	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
Contabilizar()	Responsabilidad encargada de contabilizar las liquidaciones del instrumento de cobro o pago del derecho u obligación correspondiente.
ContabilizarCobroPagoAnticipado()	Responsabilidad encargada de contabilizar los instrumentos de cobro y pago anticipados.

Nombre: ContabilizarCobroPagoAnticipado	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
Contabilizar()	Responsabilidad encargada de contabilizar los documentos de los anticipos de cobro y pago.

2.7.4 Clases del componente: Configuración

Nombre: GestmotivocancelacionController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
gestmotivocancelacionAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar motivos de cancelación y redireccionar a la interfaz.
adicionarAction ()	Responsabilidad encargada de adicionar motivos de cancelación.
modificarAction ()	Responsabilidad encargada de modificar motivos de cancelación.
eliminarmotivocancelacionAction ()	Responsabilidad encargada de eliminar motivos de cancelación.
cargarmotivocancelacionAction ()	Responsabilidad encargada de cargar los motivos de cancelación.
generaridAction()	Responsabilidad encargada de generar nuevos identificadores.

Nombre: CondicionpagoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
condicionpagoAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar la condición de pago y redireccionar a la interfaz.
cargarcondicionAction ()	Responsabilidad encargada de cargar las condiciones de pago.
adicionarcondicionAction ()	Responsabilidad encargada de guardar nuevas

	condiciones de pago.

Nombre: ParrafoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
parrafoAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar párrafos fiscales y redireccionar a la interfaz.
insertarparrafofiscalAction ()	Responsabilidad encargada de adicionar párrafos fiscales.
modificarparrafofiscalAction ()	Responsabilidad encargada de modificar párrafos fiscales.
eliminarparrafofiscalAction ()	Responsabilidad encargada de eliminar párrafos fiscales.
cargarparrafofiscalAction ()	Responsabilidad encargada de cargar los párrafos fiscales.
cargarformatoAction()	Responsabilidad encargada de cargar todos los formatos.
cargarparteformatoAction()	Responsabilidad encargada de cargar la parte de formato dado un identificador.
damelongAction()	Responsabilidad encargada de cargar la longitud del nomenclador de párrafo.

Nombre: MotivoCancelacionModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**

MotivoCancelacionModel ()	Constructor de la clase.
adicionarMotivoCancelacion ()	Responsabilidad encargada de adicionar motivos de cancelación.
eliminarMotivoCancelacion ()	Responsabilidad encargada de eliminar motivos de cancelación.

Nombre: condicionpagoModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
condicionpagoModel ()	Constructor de la clase.
adicionarcondicion ()	Responsabilidad encargada de adicionar condiciones de pago.
ModificarCondicionPago ()	Responsabilidad encargada de modificar condiciones de pago.

Nombre: parrafoModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
parrafoModel ()	Constructor de la clase.
insertarParrafoFiscal ()	Responsabilidad encargada de adicionar párrafos fiscales.
modificarParrafoFiscal ()	Responsabilidad encargada de modificar párrafos fiscales.
eliminarParrafoFiscal()	Responsabilidad encargada de eliminar párrafos fiscales.

2.7.5 Clases del componente: Carga inicial

Nombre: AperturaController
Tipo de clase: Controladora.

Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
aperturaAction()	Responsabilidad encargada de validar todos los datos necesarios para realizar la apertura inicial y redireccionar a la interfaz.
comenzarAction()	Responsabilidad encargada de realizar todo el flujo de operaciones para realizar la carga inicial.

Nombre: CobroanticipadoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
cobroanticipadoAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar los cobros anticipados y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar las monedas.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipoinstrumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargaranticiposAction()	Responsabilidad encargada de cargar los cobros anticipados.
cargaranticipoporidAction()	Responsabilidad encargada de cargar los datos de un cobro anticipado dado un id.
adicionarcobroAction()	Responsabilidad encargada de adicionar un cobro anticipado en la base de datos.
modificarcobroAction()	Responsabilidad encargada de modificar un cobro anticipado en la base de datos.

eliminarcobroAction()	Responsabilidad encargada de eliminar un cobro anticipado en la base de datos.

Nombre: PagoanticipadoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
pagoanticipadoAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar los pagos anticipados y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar las monedas.
cargarclientesAction()	Responsabilidad encargada de cargar los clientes.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipoinstrumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargaranticiposAction()	Responsabilidad encargada de cargar los pagos anticipados.
cargaranticipoporidAction()	Responsabilidad encargada de cargar los datos de un pago anticipado dado un id.
adicionarpagoAction()	Responsabilidad encargada de adicionar un pago anticipado en la base de datos.
modificarpagoAction()	Responsabilidad encargada de modificar un pago anticipado en la base de datos.
eliminarpagoAction()	Responsabilidad encargada de eliminar un pago anticipado en la base de datos.

Nombre: DerechodecobroController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**

Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
derechodecobroAction ()	Responsabilidad encargada de validar todos los datos necesarios para gestionar los derechos de cobro y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar las monedas.
cargarclientesAction ()	Responsabilidad encargada de cargar los clientes.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipodocumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargartipooperacionAction()	Responsabilidad encargada de cargar los tipos de operaciones.
cargarderechoscobroAction()	Responsabilidad encargada de cargar los derechos de cobros.
cargarderechoscobroporidAction ()	Responsabilidad encargada de cargar los datos de un derecho de cobro dado un id.
adicionarderechocobroAction ()	Responsabilidad encargada de adicionar un derecho de cobro en la base de datos.
modificarderechocobroAction ()	Responsabilidad encargada de modificar un derecho de cobro en la base de datos.
eliminarderechocobroAction ()	Responsabilidad encargada de eliminar un derecho de cobro en la base de datos.

Nombre: DerechofiscalController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
derechofiscalAction()	Responsabilidad encargada de validar todos los

	datos necesarios para gestionar los derechos fiscales y redireccionar a la interfaz.
cargarmonedasAction()	Responsabilidad encargada de cargar las monedas.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipodocumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargartipooperacionAction()	Responsabilidad encargada de cargar los tipos de operaciones.
cargarparrafofiscalAction()	Responsabilidad encargada de cargar los párrafos fiscales.
cargarderechosfiscalesAction()	Responsabilidad encargada de cargar los derechos fiscales.
cargarderechofiscalporidAction()	Responsabilidad encargada de cargar los datos de un derecho fiscal dado un id.
adicionarderechofiscalAction()	Responsabilidad encargada de adicionar un derecho fiscal en la base de datos.
modificarderechofiscalAction()	Responsabilidad encargada de modificar un derecho fiscal en la base de datos.
eliminarderechofiscalAction()	Responsabilidad encargada de eliminar un derecho fiscal en la base de datos.

Nombre: ObligaciondepagoController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
obligaciondepagoAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar las obligaciones de pagos y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar las monedas.
cargarclientesAction ()	Responsabilidad encargada de cargar los

	clientes.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipodocumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargartipooperacionAction()	Responsabilidad encargada de cargar los tipos de operaciones.
cargarobligacionespagoAction()	Responsabilidad encargada de cargar las obligaciones de pagos.
cargarobligacionpagoporidAction()	Responsabilidad encargada de cargar los datos de una obligación de pago dado un id.
adicionarobligacionpagoAction()	Responsabilidad encargada de adicionar una obligación de pago en la base de datos.
modificarobligacionpagoAction()	Responsabilidad encargada de modificar una obligación de pago en la base de datos.
eliminarobligacionpagoAction()	Responsabilidad encargada de eliminar una obligación de pago en la base de datos.

Nombre: ObligacionfiscalController.	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
obligacionfiscalAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar las obligaciones fiscales y redireccionar a la interfaz.
cargarmonedasAction ()	Responsabilidad encargada de cargar las monedas.
cargararbolcuentaAction()	Responsabilidad encargada de cargar el árbol de las cuentas.
cargartipodocumentoAction()	Responsabilidad encargada de cargar los tipos de instrumentos.
cargartipooperacionAction()	Responsabilidad encargada de cargar los tipos de operaciones.

cargarparrafofiscalAction()	Responsabilidad encargada de cargar los párrafos fiscales.
cargarobligacionesfiscalesAction()	Responsabilidad encargada de cargar las obligaciones fiscales.
cargarobligacionfiscalporidAction()	Responsabilidad encargada de cargar los datos de una obligación fiscal dado un id.
adicionarobligacionfiscalAction()	Responsabilidad encargada de adicionar una obligación fiscal en la base de datos.
modificarobligacionfiscalAction()	Responsabilidad encargada de modificar una obligación fiscal en la base de datos.
eliminarobligacionfiscalAction()	Responsabilidad encargada de eliminar una obligación fiscal en la base de datos.

Nombre: cargainicialModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
cargainicialModel ()	Constructor de la clase.
CargarMonedas ()	Responsabilidad encargada de cargar monedas.
CargarClientes ()	Responsabilidad encargada de cargar clientes.
CargarTipoDocumento ()	Responsabilidad encargada de cargar los tipos de documentos.
CargarTipoInstrumento ()	Responsabilidad encargada de cargar los tipos de instrumentos.
CargarTipoOperacion ()	Responsabilidad encargada de cargar los tipos de operaciones.
CargarParrafoFiscal ()	Responsabilidad encargada de cargar párrafos fiscales.
CargarAnticipos ()	Responsabilidad encargada de cargar los anticipos.
CargarDerechoUObligacion ()	Responsabilidad encargada de cargar los derechos u obligaciones.
CargarAnticipoPorId()	Responsabilidad encargada de cargar un anticipo dado su Id.

CargarDerechoUObligacionPorId()	Responsabilidad encargada de cargar un derecho u obligación dado su Id.
AdicionarAnticipo()	Responsabilidad encargada de adicionar anticipos.
ModificarAnticipo()	Responsabilidad encargada de modificar anticipos.
EliminarAnticipo()	Responsabilidad encargada de eliminar anticipos.
AdicionarDerechouObligacion()	Responsabilidad encargada de adicionar derechos u obligaciones.
ModificarDerechouObligacion()	Responsabilidad encargada de modificar derechos u obligaciones.
EliminarDerechouObligacion()	Responsabilidad encargada de eliminar derechos u obligaciones.
AdicionarOperacion()	Responsabilidad encargada de adicionar operaciones.
ModificarOperacion()	Responsabilidad encargada de modificar operaciones.
EliminarOperacion()	Responsabilidad encargada de eliminar operaciones.
CalcularSaldo()	Responsabilidad encargada de calcular el saldo dado el Id de la cuenta.

2.7.6 Clases del componente: Conciliación

Nombre: AdicionarconciliaciondeobligacionesoderechosController	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
adicionarconciliaciondeobligacionesoderechosAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar las obligaciones fiscales y redireccionar a la interfaz.
cargarcbtipodcAction()	Responsabilidad encargada de cargar los tipos

	de documentos.
cargarcbclientAction()	Responsabilidad encargada de cargar los clientes
cargargridAction()	Responsabilidad encargada de cargar los datos de los derechos u obligaciones, dado el cliente, el tipo de documento, fecha inicio y fecha final.
adicionarconciliacionAction()	Responsabilidad encargada de adicionar las conciliaciones

Nombre: AdicionarconcilaciondeanticiposController	
Tipo de clase: Controladora.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
init()	Constructor de la clase.
adicionarconcilaciondeanticiposAction()	Responsabilidad encargada de validar todos los datos necesarios para gestionar las obligaciones fiscales y redireccionar a la interfaz.
cargarcbtipodocAction()	Responsabilidad encargada de cargar los tipos de documentos.
cargarcbclienteAction()	Responsabilidad encargada de cargar los clientes
cargargridanticipoAction()	Responsabilidad encargada de cargar los datos de los anticipos, dado el cliente, el tipo de documento, fecha inicio y fecha final.
adicionarconciliacionaAction()	Responsabilidad encargada de adicionar las conciliaciones

Nombre: DatConciliacionModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**

Para cada responsabilidad	
Nombre.	**Descripción.**
DatConciliacionModel ()	Constructor de la clase.
insertarConciliacion ()	Responsabilidad encargada de adicionar una conciliación.

Nombre: DatRefanticipoModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
DatRefanticipoModel ()	Constructor de la clase.
insertrefanticipo ()	Responsabilidad encargada de adicionar una referencia a una conciliación de anticipos.

Nombre: DatRefderechouobligacionModel.	
Tipo de clase: Entidad.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre.	**Descripción.**
DatRefderechouobligacionModel ()	Constructor de la clase.
insertrefderecho ()	Responsabilidad encargada de adicionar una referencia a una conciliación de obligación o derechos.

2.8 Conclusiones del Capítulo 2

Con la realización de este capítulo, se arribó a la conclusión, que la obtención del diseño propuesto por el analista resultó muy importante, pues permitió adquirir una comprensión de todo lo relacionado con los requisitos no funcionales y con las restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.

Además, al aplicar estilos de código se hace más legible el programa fuente, lo cual ayuda en la comunicación entre desarrolladores, y permite una temprana detección de errores. La propuesta ofrece estándares de codificación, por los que se rigen los programadores del proyecto, son completamente extensibles. Queda de parte de quienes la extiendan tratar de mantener un equilibrio, permitiendo una codificación homogénea y evitar las redundancias.

También resultó importante la descripción de uno de los algoritmos no triviales tratando de lograr que se entienda como fue pensado y desarrollado el mismo. De la misma forma quedaron descritas las estructuras de datos que serán necesarias para la implementación del módulo, y las clases más importantes definidas; tales como las clases controladoras y entidad.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

Durante el proceso de desarrollo de un software, los errores pueden comenzar a aparecer incluso desde el mismo momento en que fueron definidos los objetivos y estos a su vez especificados de forma errónea e imperfecta; de la misma forma en los posteriores pasos del diseño y desarrollo. A raíz de la incapacidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software debe ser acompañado de una actividad que garantice su calidad.

La aplicación de métricas y pruebas de software constituyen un elemento crítico para la garantía de la calidad del software. Además el uso de las pruebas representa una revisión final de las especificaciones del diseño y de la codificación.

La creciente inclusión del software, como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo, han motivado la creación de pruebas más minuciosas y bien planificadas.

3.2 Métricas

Un aspecto importante a tener en cuenta en la evaluación de la calidad del diseño ha sido la aplicación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos referenciadas por Pressman [24]; teniendo en cuenta que este estudio brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software.

Atributos de calidad que se abarcan:

1. **Responsabilidad:** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
2. **Complejidad del diseño:** Consiste en la complejidad que posee una estructura de diseño de clases.
3. **Complejidad de implementación:** Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
4. **Reutilización:** Consiste en el grado de reutilización que presente una clase o estructura de clase, dentro de un diseño de software.
5. **Acoplamiento:** Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, esta muy ligada a la característica de Reutilización.
6. **Complejidad del mantenimiento:** Consiste en el 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.

7. **Cantidad de pruebas:** Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (Componente, modulo, clase, conjunto de clases, etc.) diseñado.
8. **Nivel de Cohesión:** Consiste en el grado de especialización de las clases concebidas para modelar un dominio o concepto específico.
9. **Abstracción del diseño:** Consiste en la capacidad de modelar lo más cercano posible a la realidad un concepto o dominio determinado.

Las métricas aplicadas como instrumento para evaluar la calidad del diseño del módulo Cobros y Pagos y su relación con los atributos de calidad definidos en este trabajo son las siguientes:

Tamaño operacional de la clase (TOC): Está dado por el número de métodos asignados una clase.

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Tabla 3: Tamaño operacional de la clase (TOC)

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otras.

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 4: Relaciones entre clases (RC)

3.2.1 Resultados del instrumento de evaluación de la métrica Tamaño operacional de la clase (TOC).

Ver instrumentos y tabla de resultados en (Anexo 14 Instrumento de medición de la métrica Tamaño operacional de la clase (TOC)).

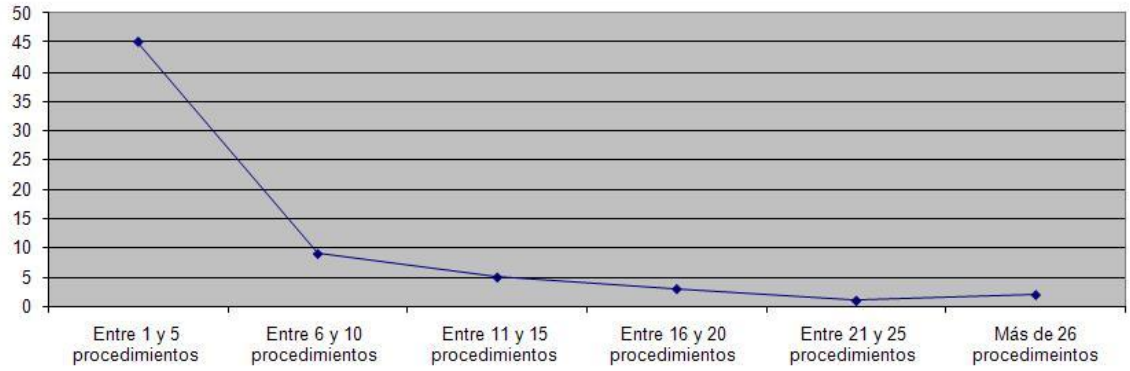


Figura 5: Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

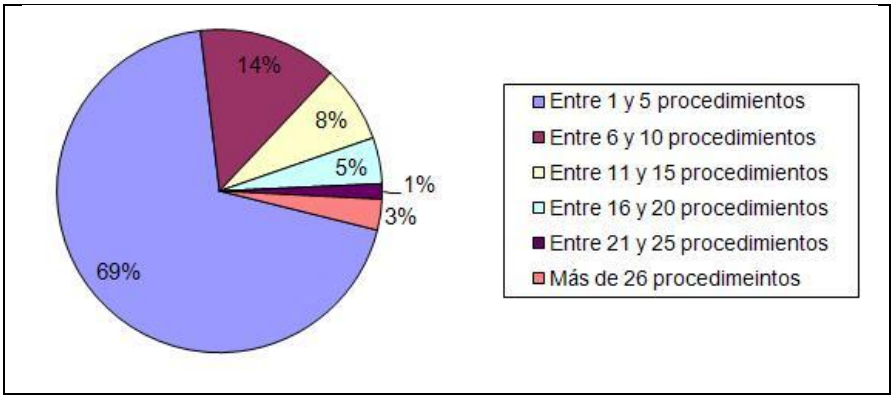


Figura 6: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

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

Figura 8: 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 TOC, se puede concluir que el diseño del módulo Cobros y Pagos tiene una calidad aceptable teniendo en cuenta que el 97% de las clases incluidas en estos subsistemas posee menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones. Además el 89% de las clases poseen evaluaciones positivas en los atributos de calidad (Complejidad de Implementación y Reutilización).

3.2.2 Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC).

Ver instrumentos y tabla de resultados en (Anexo 15 Instrumento de medición de la métrica Relaciones entre clases (RC)).

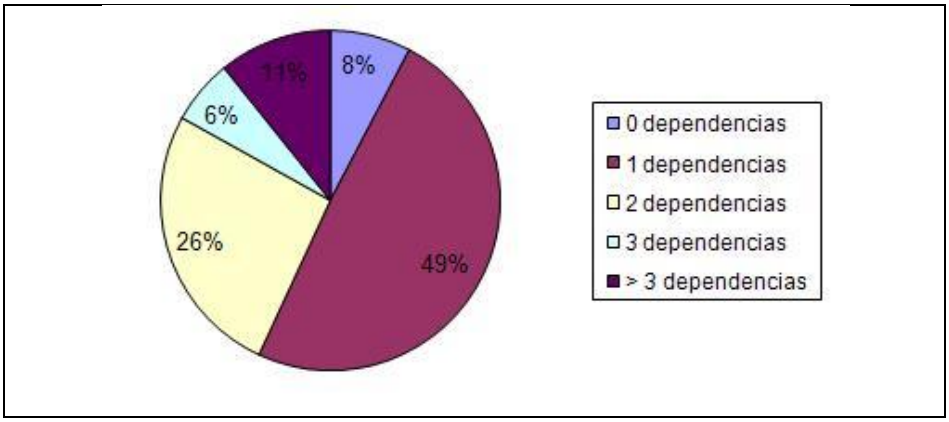


Figura 9: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

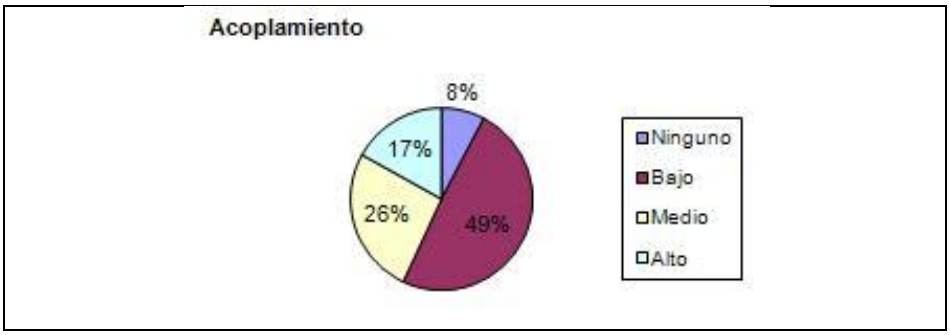


Figura 10: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

Figura 11: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

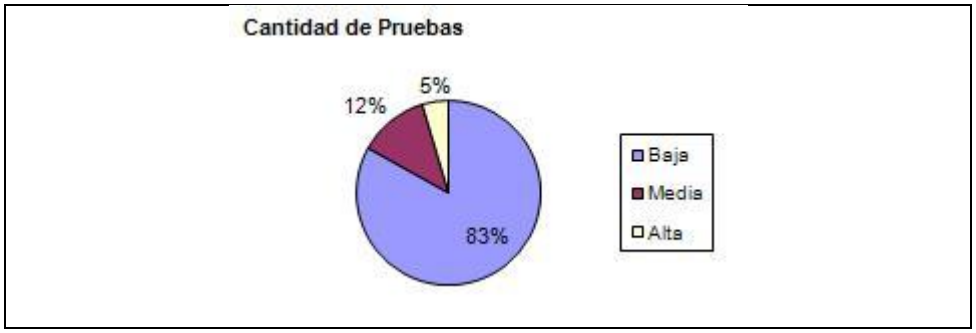


Figura 12: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

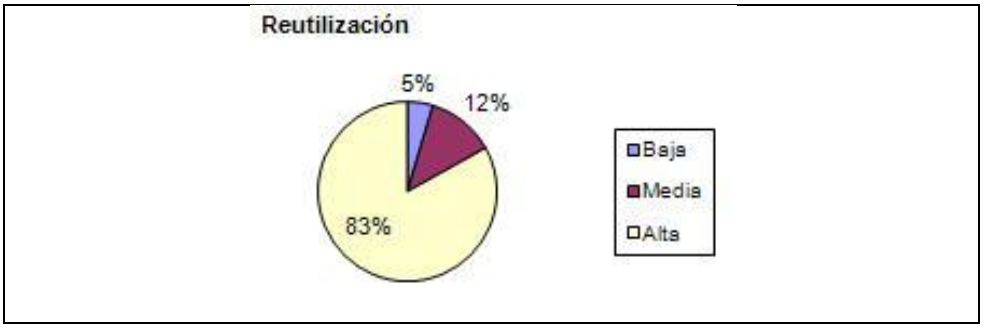


Figura 13: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del módulo Cobros y Pagos tiene una calidad aceptable teniendo en cuenta que el 83% de las clases incluidas en el módulo posee menos de 3 dependencias de otras clases. Además el 8% de las clases no poseen acoplamiento con otras y el 83% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 95% de las clases.

A manera de resumen se han tabulado los resultados obtenidos en la siguiente tabla.

Tabla 8: Resumen de los resultados

Atributos de Calidad	TOC	RC	Prom.	Eval.
Complejidad de Implementación	1		1	B
Reutilización		1	1	B
Acoplamiento		1	1	B
Complejidad de Mantenimiento.		1	1	B
Cantidad de Pruebas		1	1	B
Abstracción	1		1	B
Cohesión	1		1	B

Leyenda:

0.1 a 0.3	M
0.4 a 0.7	R
0.8 a 1	B

3.3 Pruebas de Software

Las pruebas son el proceso de ejercitar un programa con la intención específica de encontrar errores previos a la entrega al usuario final. Las pruebas son la actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, donde los resultados son observados y registrados, y es hecha una evaluación de algún aspecto del sistema o componente.

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

Al concluir la prueba se evalúan los resultados obtenidos frente a los resultados esperados. Si se descubren datos erróneos implica que hay un error y hay que

corregirlo y empieza el proceso de depuración de errores. Se basa en las estructuras de control del diseño procedimental para generar los casos de prueba que:

- ✓ Garanticen que se recorran por lo menos una vez todos los caminos independientes de cada módulo.
- ✓ Se ejecutan todas las decisiones lógicas en su parte verdadera y en su parte falsa.
- ✓ Se recorren todos los bucles.
- ✓ Se utilizan las estructuras de datos internas para garantizar su validez.
- ✓ Se invierte tiempo y esfuerzo en los detalles de control debido a que:
- ✓ Los errores suelen estar en situaciones fuera de las normales.
- ✓ A menudo caminos que se piensa que tienen pocas posibilidades de recorrerse, son recorridos regularmente.
- ✓ Los errores tipográficos son aleatorios. Puede que no sean detectados por los procesadores de la sintaxis del lenguaje particular y estar presentes en cualquier camino lógico.

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

3.4 Descripción de los test de Unidad

Los “test de unidad” son pruebas llevadas a cabo por los implementadores sobre las unidades mínimas desarrolladas por ellos, estas unidades pueden ser clases, métodos, propiedades, componentes, etc. Estas unidades se prueban separadas unas de otras y básicamente se hacen durante la implementación del software.

En este nivel de prueba fundamentalmente se ejecutan casos de pruebas de caja blanca diseñados para:

- ✓ Probar las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente, conservan su integridad durante todos los pasos de ejecución del algoritmo.
- ✓ Probar las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.
- ✓ Ejercitar todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y, finalmente, se prueban todos los caminos de manejo de errores.

Los casos de pruebas diseñados para ejecutar a nivel de unidad deben descubrir errores como:

- ✓ Comparaciones entre tipos de datos distintos.
- ✓ Operadores lógicos o de procedencia incorrectos.
- ✓ Igualdad esperada cuando los errores de precisión la hacen poco probable.
- ✓ Variables o comparaciones incorrectas.
- ✓ Terminación de bucles inapropiada o inexistente.
- ✓ Fallo de salida cuando se encuentra una iteración divergente.
- ✓ Variables de bucles modificadas de forma inapropiada.

La prueba de límites es probablemente la más importante. El software falla con frecuencia en sus condiciones límites. Las pruebas que ejercitan las estructuras de datos, el flujo de control y los valores de los datos por debajo y por encima de los máximos y los mínimos son muy apropiadas para descubrir estos errores. [24]

Tipos de pruebas:

***Técnicas de caja blanca o estructurales*:** que se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

***Técnicas de caja negra o funcionales*:** que realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.

Ver *Anexo 7*

3.4.1 Pruebas de Caja Blanca

Consiste en realizar pruebas para verificar que líneas específicas de código funcionan tal como está definido. También se le conoce como prueba de caja-transparente. Esta se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar. [23]

Estas tienen como objetivos:

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

- ✓ Ejercitar las estructuras internas de datos para asegurar su validez.

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

Entre las técnicas de prueba de Caja Blanca podemos ver:

1. **Prueba de Condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
2. **Prueba de Flujo de Datos:** Se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
3. **Prueba de Bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.
4. **Prueba del Camino Básico:** Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

Nos detendremos en las pruebas de camino básico, la esencia de las mismas es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática.

Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

3.4.2 Pruebas de Caja Negra

La prueba de Caja Negra se centra principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos. [23]

La misma no es una alternativa a las técnicas de prueba anteriormente vistas, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.
Estas pruebas permiten encontrar:

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

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

Para esta prueba, existen varias técnicas, entre ellas están:
1. Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

2. Técnica del Análisis de Valores Límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Dentro del método de Caja Negra la técnica de la Partición de Equivalencia es una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar. En este apartado se hará uso de esta técnica para aplicarle a una funcionalidad del sistema desarrollado el método de caja negra.

3.5 Aplicación de pruebas de caja blanca

Para realizar el test es necesario realizar primeramente los procesos descritos en el capítulo 2, epígrafe 2.5.1 “Análisis de complejidad del algoritmo” para calcular los valores de la complejidad ciclomática del procedimiento al cual se le va a aplicar la prueba. A continuación se enumera las sentencias de código del procedimiento.

```php
public function CargarGridAction()
{   //1
    $entidad = $this->global->Estructura->idestructura; //1
    $idtipo = $this->_request->getPost('idtipoa'); //1
    $idcliente= $this->_request->getPost('cliente'); //1
    $desde= $this->_request->getPost('dfdesde'); //1
    $hasta= $this->_request->getPost('dfhasta'); //1
    $arridderecho= $this->pIntegrator->cobroPago->DeruObConContratosNoContabilizados($entidad); //1

    $valor = array('arrayid'=>$arridderecho,'entidad' => $entidad, 'idtipo' => $idtipo,
                   'idcliente' => $idcliente, 'fini' => $desde, 'ffin' => $hasta); //1
    $result = $this->pIntegrator->cobroPago->getDerOb($valor); //1
    $cant = count($result); //1

    for($i=0; $i<$cant; $i++) //2
    {
        $cond[] = $this->pIntegrator->cobroPago->ObtenerCondicionPago($result[$i][idcondicionpago]); //3
        $todo[] = array('numero' => $result[$i]['numero'],  //3
        'fechaemision' => $result[$i]['fechaemision'], //3
        'plazo' => $cond[$i][0]['plazo']); //3
    }
    die(json_encode(array('grid' => $todo))); //4
} //5
```

Figura 23: Representación del algoritmo CargarGridAction()

Seguidamente se construye el grafo de flujo asociado al código anterior.

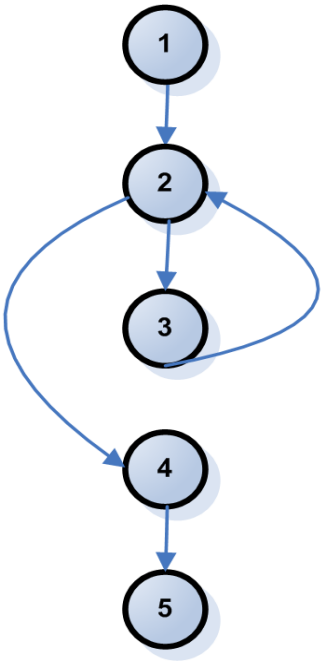


Figura 24: Grafo de flujo asociado al algoritmo CargarGridAction()

Seguidamente se realiza el cálculo de la complejidad ciclomática, mediante las fórmulas descritas en el capítulo 2, epígrafe 2.5.1 “Análisis de complejidad del algoritmo”, las cuales tienen que arrojar el mismo resultado para asegurar que el cálculo de la complejidad es correcto.

Fórmulas para calcular complejidad ciclomática.

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

Aplicando estas fórmulas al grafo de flujo de la figura 5 se obtienen los siguientes resultados:

Calculando mediante la fórmula 1:

V (G) = (5 - 5) + 2

V (G) = 2

Calculando mediante la fórmula 2:

V (G) = 1 + 1

V (G) = 2

Calculando mediante la fórmula 3:

V (G) = 2

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 2, lo que significa que existen dos posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes a los demás.

Camino básico #1:

1 – 2 – 4 – 5

Camino básico #2:

1 – 2 – 3 – 2 – 4 – 5

Una vez extraídos los caminos básicos del flujo, se debe realizar al menos un caso de prueba por cada camino básico.

Para realizar los casos de pruebas es necesario cumplir con las siguientes exigencias:

Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que entran al procedimiento

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

Caso de prueba para el camino básico # 1.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El id del cliente, el id tipo de documento son números enteros; las fechas de inicio y final están en formato de fecha.

Condición de ejecución: El id del cliente será igual a 2, el id del tipo de documento será igual 1; la fecha de inicio 05/03/2009 y la fecha de final 05/04/2009.

Entrada: $idcliente = 2, $idtipo = 1, $desde = 05/03/2009, $hasta = 05/04/2009.

Resultados esperados: Se espera que se muestren los datos en el grid.

Resultados: No hay información en la BD que coincida con esos parámetros de entrada, los datos no se muestran en el grid.

Caso de prueba para el camino básico # 2.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El id del cliente, el id tipo de documento son números enteros; las fechas de inicio y final están en formato de fecha.

Condición de ejecución: El id del cliente será igual a 1, el id del tipo de documento será igual 1; la fecha de inicio 03/02/2009 y la fecha de final 05/03/2009.

Entrada: $idcliente = 1, $idtipo = 1, $desde = 03/02/2009, $hasta = 05/03/2009.

Resultados esperados: Se espera que se muestren los datos en el grid.

Resultados: En la BD se encuentran datos coincidentes con los parámetros de entrada, los datos se muestran en el grid.

Concluidos los casos de pruebas fue posible verificar que el flujo de trabajo en la función esta correcto, puesto que cumple con las condiciones necesarias planteadas.

3.6 Aplicación de pruebas de caja negra

Para el desarrollo de este tipo de prueba se utilizará el requisito Adicionar Conciliación de Anticipos. El objetivo del mismo es persistir en la BD la información referente a los pagos o cobros anticipados que han sido conciliados por la entidad y dando la posibilidad de cancelar o no un anticipo.

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Adicionar Conciliación de Anticipos.	El sistema permite adicionar o realizar la conciliación de uno o varios anticipos, persistiendo la información en la BD, y dando la posibilidad también de cancelar o no dichos anticipos.	EP 1.1: Buscar los anticipos posibles a conciliar o no, introduciendo los datos correctamente al presionar el botón **Aceptar.**	– Se presiona el botón **Buscar.** – Se introducen los datos correctamente. – Se presiona el botón **Aceptar.** – Se muestra la información de los anticipos que coinciden con los parámetros de búsqueda.
		EP 1.2: Buscar los anticipos posibles a conciliar o no, introduciendo los datos inválidos al presionar el botón **Aceptar.**	– Se presiona el botón **Buscar.** – Se introducen los datos correctamente. – Se presiona el botón **Aceptar.** – Se muestra la notificación “No existen anticipos con los parámetros de su búsqueda”. – Se presiona el botón **Aceptar** del mensaje.

		EP 1.3: Buscar los anticipos posibles a conciliar o no, dejando campos requeridos en blanco al presionar el botón **Aceptar.**	– Se presiona el botón **Buscar**. – Se introducen los datos correspondientes en el formulario y se deja al menos un campo requerido en blanco. – Se presiona el botón **Aceptar.** – Se muestra la notificación “Debe completar todos los campos para la búsqueda”. – Se presiona el botón **Aceptar** del mensaje.
		EP 1.4: Cancelar.	– Se presiona el botón **Buscar**. – Se introducen o no los datos en los campos. – Se presiona el botón **Cancelar**.
		EP 1.5: Adicionar una Conciliación de anticipos introduciendo los datos correctamente al presionar el botón **Aceptar.**	– Se marca el anticipo a conciliar – Se le aplican los cambios correctamente. – Se presiona el botón **Aplicar.** – Se muestran los cambios en la interfaz. – Se presiona el botón **Aceptar.** – Se muestra la notificación “La conciliación ha sido

			registrada satisfactoriamente". – Se presiona el botón **Aceptar** del mensaje.
		EP 1.6: Adicionar una Conciliación de anticipos dejando campos requeridos en blanco al presionar el botón **Aceptar.**	– Se dejan anticipos sin definir el campo "coincide". – Se presiona el botón **Aceptar.** – Se muestra la notificación "Todos los anticipos deben contener sus campos completos". – Se presiona el botón **Aceptar** del mensaje.

3.7 Conclusiones del Capítulo 3

En este capítulo se abordaron diferentes aspectos relacionados con métricas aplicadas al diseño y las pruebas de software, se realizó una descripción de las métricas aplicadas al diseño, así como de los test de unidad, entre los cuales están las pruebas de Caja Blanca y Caja Negra, quedando demostrado que las pruebas al software constituyen una herramienta más para que el mismo tenga la calidad requerida antes de ser entregado al cliente, y este quede satisfecho con el trabajo realizado.

CONCLUSIONES

Al concluir el presente trabajo, se confirma la necesidad y la importancia de contar con un módulo que informatice los procesos de cobros y pagos en las entidades del país; ayudando a resolver de forma eficiente los problemas referentes a la gestión empresarial en nuestras entidades. Además ha representado para los autores la posibilidad de aplicar los conocimientos adquiridos durante los cinco años de la carrera.

Al finalizar este trabajo se ha logrado cumplir el objetivo trazado pues se realizó un estudio de sistemas informáticos vinculados con la gestión de los procesos de cobros y pagos; se llevó a cabo un estudio de las herramientas, tecnologías y estándares de codificación definidos por la línea de Arquitectura, lo cual posibilitó entender las causas de su elección, constituyendo estándares y políticas precisas para la producción de software en el país. Se hizo el estudio del modelado del diseño realizado por las analistas. Además se aplicaron métricas y realizaron diferentes tipos de pruebas con el objetivo de comprobar la viabilidad de la solución.

En resumen, se ha cumplido el objetivo de la investigación, pues se realizó la implementación del módulo Cobros y Pagos del Sistema Integral de Gestión Cedrux utilizando los patrones de diseños establecidos, además de una interfaz gráfica orientada al usuario, agradable y fácil de operar; lográndose abarcar las funcionalidades previstas, facilitando así el trabajo de los funcionarios encargados de controlar dichos procesos en las empresas donde sea desplegada la aplicación.

RECOMENDACIONES

Con la realización del presente trabajo se recomienda:

- Incorporar una funcionalidad que permita visualizar el comprobante de operaciones que se genera dentro del módulo, eliminando la necesidad de ir hasta el Subsistema de Contabilidad para tener acceso al mismo.

- Continuar incorporando funcionalidades que hoy se desarrollan en el módulo Cobros y Pagos al componente ComúnFinanzas del Subsistema de Finanzas, con el objetivo de lograr una mayor y mejor integración entre nuestro módulo y el resto del Subsistema de Finanzas del Sistema Integral de Gestión Cedrux.

- Comenzar el proceso de implantación del módulo en las entidades piloto con el objetivo de trabajar en las incidencias provenientes del mismo e ir perfeccionando el trabajo realizado.

BIBLIOGRAFÍA

[1] *¿Qué es un ERP?* .p Disponible en:

http://www.adpime.com (20/2/2009).

[2] CUEVAS, G. E. G. *Tecnología de la información como herramienta para aumentar la productividad de una empresa.*

[3] OCHOA, E. S. *Ventajas y Desventajas de ERP*. 2007. p. Disponible en: http://secretosenred.com/articles/2753/2751/VENTAJAS-Y-DESVENTAJAS-DE-ERP/Paacutegina2751.html (20/2/2009).

[4] ZARAGOZA, C. *OpenBravo*. p. Disponible en: http://www.openbravo.com/docs/openbravo-2.3-release-aug-03-2007-esp.pdf (20/2/2009).

[5] *SAP número uno en Software ERP*. p. Disponible en: http://www.sap.com/spain/solutions/business-suite/erp/index.epx (20/2/2009).

[6] TAMARGO, L. *La contabilidad en una nueva tecnología*. P. Disponible en: http://www.betsime.disaic.cu/secciones/tec_feb_02.htm (20/2/2009).

[7] MARTÍN, E. L. *Un efectivo sistema cubano de contabilidad*. 2005, p. Disponible en: http://www.eleconomista.cubaweb.cu/2005/edicionimpresa/premioeconomiamiguel.html. (20/2/2009).

[8] HIDALGO, I. C. *Sistema de gestión Integral ASSETS-NS: garantía de seguridad total.* 2004, p. Disponible en:

http://www.cadenagramonte.cubaweb.cu/ciencia/sistemas_gestion_integral.asp. (20/2/2009).

[9] PRODUCCIÓN, *Equipo de. Modelo de Desarrollo Orientado a Componentes ERP-Cuba.* Ciudad Habana : s.n., 2009.

[10] *Herramientas Web para la enseñanza de Protocolos de Comunicación. El modelo cliente – servidor.* Disponible en:

http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/cliente-servidor.html (20/2/2009).

[11] *Introducción a las Aplicaciones Web*. Disponible en:

http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node11.html (20/2/2009).

[12] DE LA TORRE, Aníbal. *Lenguajes del lado del Servidor o Cliente,* 2006. Disponible en: http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html (20/2/2009).

[13] *Extjs*. p Disponible en: http://webmaster-mexico.com/ (20/2/2009).

[14] *Lenguajes de Programación para páginas Web HTML,* 2007. Disponible en:

http://www.monografias.com/trabajos7/html/html.shtml (20/2/2009).

[15] DE LA TORRE, Aníbal. *Lenguajes del lado del Servidor o Cliente,* 2006. Disponible en: http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html (20/2/2009).

[16] *Lenguajes de Programación que deberías aprender,* 2007. Disponible en:

http://www.tufuncion.com/diferentes-lenguajes-programacion (20/2/2009).

[17] TORRES, Ariel. *Que podemos hacer con PHP?,* 2007. Disponible en:

http://www.articuloweb.com/category.php?cat_id=3 (20/2/2009).

[18] El servidor de web Apache: Introducción práctica. Disponible en:

http://acsblog.es/articulos/trunk/LinuxActual/Apache/html/index.html (20/2/2009).

[19] *¿Qué es Postgres?* .p

http://www.postgres-gl.com.ar/html/informacion.php?opcion=queespostgres (20/2/2009).

[20] Zend Studio. p Disponible en:

http://www.desarrolloweb.com/programas/ (20/2/2009).

[21] COLECTIVO DE AUTORES. El patrón de diseño Proxy, 2009.

[22] GUTIERREZ, J. A. S. PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad), 2007. Disponible en:

http://jorgesaavedra.wordpress.com/tag/patrones-grasp

[23] RIZZI, I. F. M. COMPLEJIDAD CICLOMÁTICA, 2006. [2/3/2009]. Disponible en: http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZICOMPLEJIDAD.Pdf

[24] PRESSMAN, R. Ingeniería del Software: Un enfoque práctico. Madrid: McGraw Hill Prentice-Hall, 2001.

[25] Resolución 235 del 2005 del Ministerio de Finanzas y Precios (Normas Cubanas de Contabilidad).

[26] Resolución 297 del 2003 del Ministerio de Finanzas y Precios (Normas de Control Interno).

[27] Resolución 294 del 2005 del Ministerio de Finanzas y Precios (Mofication a la Resolución 235 del 2005).

GLOSARIO DE TÉRMINOS

Cobros y Pagos: Pagos son las salidas de dinero y cobros las entradas que se produzcan en la tesorería de la empresa. Así, en principio, ante el alta o la baja de algún elemento patrimonial, la empresa paga o cobra dinero.

Cobro Anticipado: Es el efectivo recibido de clientes sin que haya mediado la contraprestación del servicio o la entrega del producto o de la mercancía.

Contador: Se le llama a la persona encargada de realizar y controlar los procesos contables en una entidad.

Cuenta: Registro que contiene las operaciones de igual naturaleza y la fecha de cada una de ellas en orden cronológico. Las anotaciones en una cuenta se expresan en débitos y créditos, evaluados en términos monetarios, y muestran el saldo actual en caso de existir.

DBAL: Es una capa de abstracción de bases de datos, es decir, una interfaz de programación de aplicaciones que unifica la comunicación entre una aplicación informática y bases de datos tales como PostgreSQL

Derecho de cobro: Es un derecho que tiene el acreedor frente al deudor de cobrar lo debido por venta de mercancías o prestación de servicios.

Derechos Fiscales: Representan los importes pendientes de recibir del presupuesto del Estado por los pagos en exceso por concepto de impuestos, contribuciones, y de pagos por concepto de seguridad social a corto plazo, pendientes de reintegrar.

Dom: Document Object Model (una traducción al español no literal, pero apropiada, podría ser Modelo en Objetos para la representación de Documentos), abreviado DOM, es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML.

Eclipse: es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Entidad (empresarial): Una unidad económica que realiza transacciones comerciales que se deben registrar, resumir y reportar. Se considera la entidad separada de su propietario o propietarios.

Entidades piloto: Se le denomina a las entidades escogidas para probar el software en las condiciones reales de trabajo.

IoC: Inversión de Control del inglés Inversion of Control.

Json: acrónimo de "JavaScript Object Notation", es un formato ligero para el intercambio de datos.

Módulo: Es un componente auto-controlado de un sistema, el cual posee una interfaz bien definida hacia otros componentes; algo es modular si es construido de manera tal que se facilite su ensamblaje, acomodamiento flexible y reparación de sus componentes.

Obligaciones Fiscales: Representan los importes pendientes de abonar al presupuesto del Estado por concepto de impuestos, contribuciones y pagos por concepto de seguridad social a corto plazo pendientes de reintegrar.

Obligaciones de Pago: Representan los importes de pago a proveedores por operaciones corrientes, independientemente que su pago se efectúe previa o posteriormente a la recepción o aceptación de las mercancías, materiales y servicios recibidos.

ORM: El Mapeo Objeto Relacional es una técnica de programación para 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.

Pago Anticipado: Son los pagos efectuados a los proveedores por la recepción futura de los productos o mercancías, así como por la aceptación posterior de los trabajos o servicios, en virtud de los contratos suscritos o pactos aceptados.

Sistema: Es un conjunto de elementos interrelacionados e interdependientes de forma tal que dan origen a una unidad conceptual compleja. Posee un objetivo básico o fundamental que puede lograrse con la interrelación de los elementos.

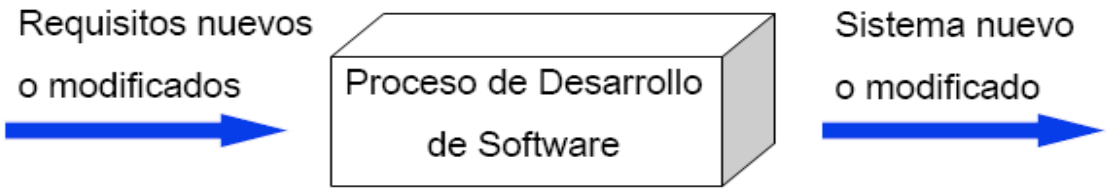
SSL: Protocolo de comunicación de datos desarrollado por Netscape para transmitir documentos privados a través del Internet.

Software: Programas de sistema, utilerías o aplicaciones expresados en un lenguaje de máquina.

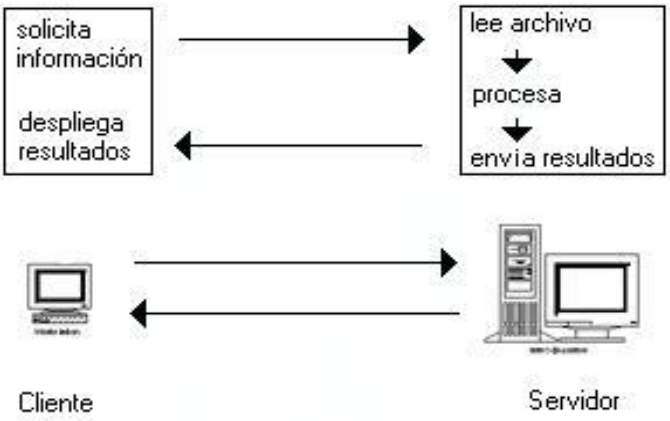
Unidades presupuestadas: Son las entidades que reciben del Estado los recursos destinados a financiar los diferentes servicios: educación, salud, actividades socioculturales, científicas, administrativas, defensa y orden interior debiendo mantener el control de los mismos, son aquellas que se dedican a recibir, ejecutar y controlar los recursos financieros que el estado brinda para financiar sus actividades.

ANEXOS

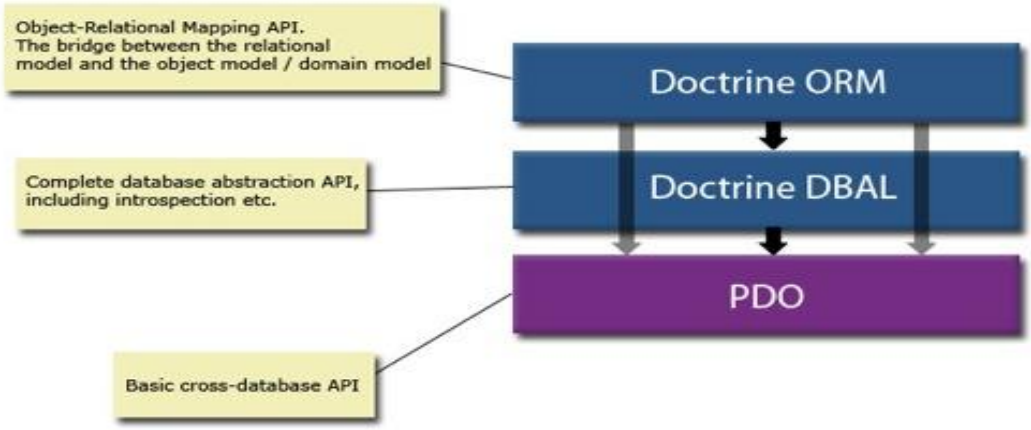
Anexo 1. Entrada y salida de un proceso de desarrollo de software



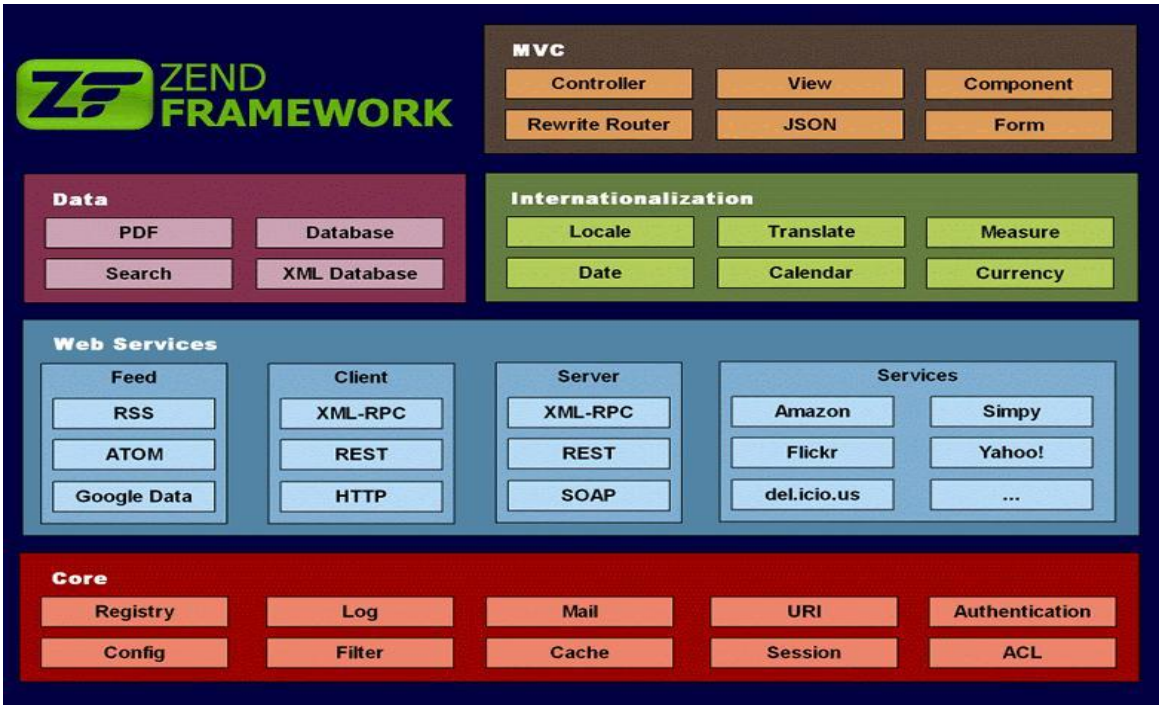
Anexo 2. Arquitectura Cliente Servidor



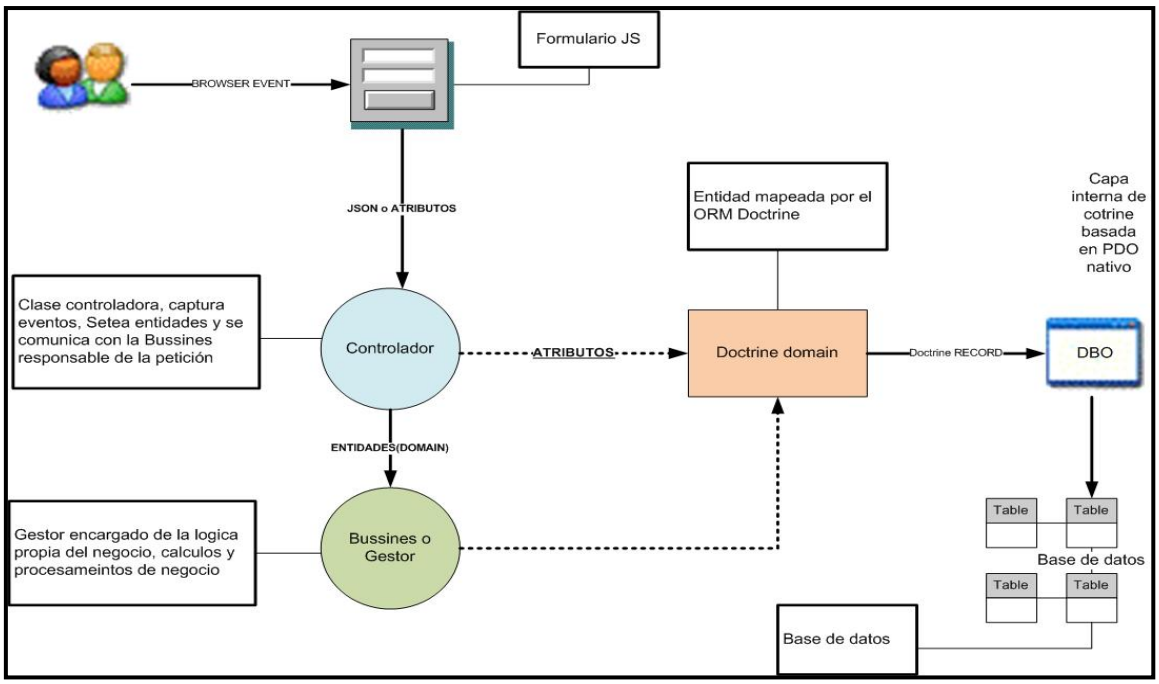
Anexo 3. Doctrine ORM



Anexo 4. Representación del Zend Framework



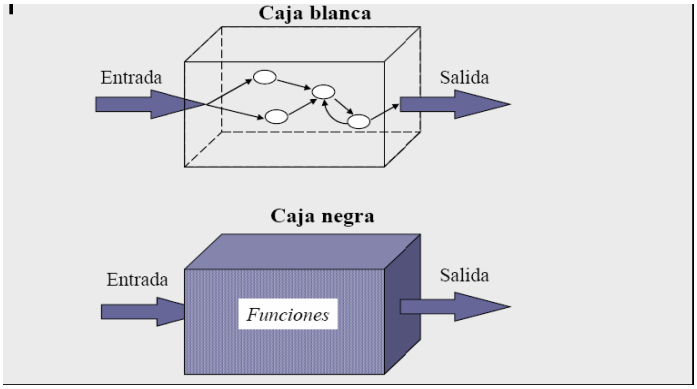
Anexo 5. Colaboración entre clases en la arquitectura



Anexo 6. Código con sus instrucciones enmarcadas

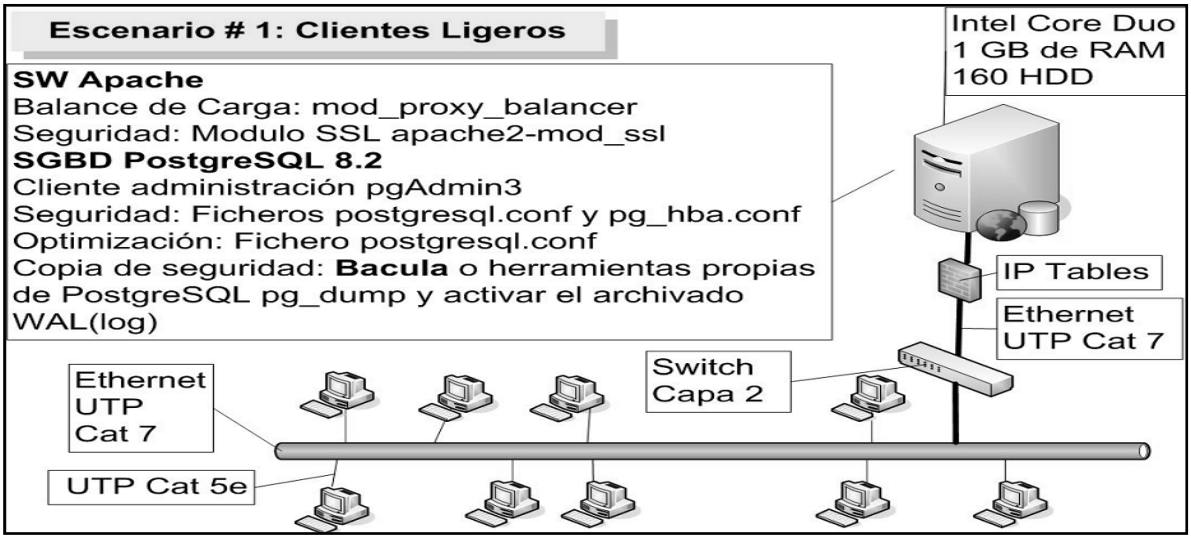
```
public function adicionarconciliacionAction()
{ //1
    $fecha=$this->_request->getPost('fecha');   //1
    $array=(array)json_decode(stripslashes($this->_request->getPost('select'))); //1
    $arrayiderecho = array(); //1
    $arrayiddoc = array(); //1
    $iddocumento = 0; //1
    foreach($array as $i=>$obj) //2
    {
        $coin=$obj->coincide; //3
        $chek=$obj->operacion; //3
        $plazo=$obj->plazo; //3
        $num=$obj->numero;  //3
        $idEstructuraComun=$this->global->Estructura->idestructura; //3
        $idsubsistema=$this->global->Subsistema->idsubsistema; //3
        $idperiodo = $this->integrator->parametros->PeriodoActualSubsistema($idEstructuraComun,$idsubsistema); //3

        $obj_conc = new DatConciliacion();  //3
        $obj_conc->fecha = $fecha; //3
        $obj_conc->idusuario=1; //3
        $obj_conc->idestructuracomun=$idEstructuraComun;  //3
        $obj_conc->idperiodo= $idperiodo;    //3

        if($coin=='Si') //4
        $obj_conc->coincide=1; //5
        else
        $obj_conc->coincide=0; //6

        if($chek=='Cancelar') //7
        $obj_conc->idoperacionconciliacion=2; //8

        else if($chek=='Cambio plazo') //9
        $obj_conc->idoperacionconciliacion=1; //10

        else
        $obj_conc->idoperacionconciliacion=3;   //11

        DatConciliacionModel::insertarConciliacion($obj_conc); //12
        $obj_ref=new DatRefderechoubligacion(); //12
        $idrecho=$this->pIntegrator->cobroPago->ObtenerIdDerecho($num);     //12
        $obj_ref->idderechoubligacion=$idrecho[0]['idderechoubligacion'];  //12
        $obj_ref->idconciliacion=$obj_conc->idconciliacion;         //12
        DatRefderechoubligacionModel::insertrefderecho($obj_ref); //12

        if($coin=='Si' && $chek=='Cambio plazo') //13
            {
                $idcondp=$this->pIntegrator->cobroPago->ObtenerIdCondicionPago($num);   //14
                $objeto=$this->pIntegrator->cobroPago->ObtenerCondicionPago($idcondp[0]['idcondicionpago']); //14
                $dpp=$objeto[0]['dpp'];  //14
                $mora=$objeto[0]['mora'];   //14
                $activo=$objeto[0]['activo'];   //14
                $cambio=$objeto[0]['idcondicionpago'];  //14
                $idestructurac=$objeto[0]['idestructuracomun'];  //14

                $idnew=$this->pIntegrator->cobroPago->InsertarCondicionPago($dpp,$mora,$plazo,$activo,$cambio,$idestructurac); //14
                $this->pIntegrator->cobroPago->ModificarActivoCondicionPago($idnew); //14
                $this->pIntegrator->cobroPago->ModificarCondicionPagoDerecho($idrecho[0]['idderechoubligacion'],$idnew); //14
            }
        elseif ($coin=='No' && $chek=='Cancelar') //15
            {
                $arrayiderecho []    = $idrecho[0]['idderechoubligacion'];  //16
                if ($iddocumento == 0) //17
                {
                    $tipodocbyalias = $this->integrator->parametros->TipoDocumentoByAlias('cambiocuenta');  //18
                    $iddocumento = $this->pIntegrator->cobroPago->InsertarDocumentoConc($fecha,$idperiodo,$idEstructuraComun,$tipodocbyalias->iddoc); //18
                }
            }
    }
    if ($iddocumento != 0) //19
    {
        $arrayiddoc [] = $iddocumento; //20
    }
    $arrayoperbyalias = $this->integrator->parametros->TipoOperacionesDoc('cambiocuenta'); //21
    $cantoper = count($arrayoperbyalias);     //21
    if($cantoper > 0)  //22
    {
        for($i=0; $i<$cantoper; $i++)   //23
        {
            $objoper = $arrayoperbyalias[$i];  //24
            if($objoper->denominacion == 'litigio')  //24
            $tipooperbyalias = $objoper->idoperacion;  //24
        }
    }
    $canti = count($arrayiderecho);  //25
    if($canti > 0)     //26
    {
        for($i=0; $i<$canti; $i++) //27
        {
            $this->pIntegrator->cobroPago->InsertarCambioCuenta($arrayiderecho[$i], $iddocumento, $tipooperbyalias); //28
            $this->pIntegrator->cobroPago->CambiarEstadoCanceladoDerecho($arrayiderecho[$i]); //28
        }
    }
    $cantiddoc = count($arrayiddoc); //29
    if($cantiddoc > 0) //30
    {
        $this->pIntegrator->cobroPago->ContabilizarCambiodeCuentaDO($arrayiddoc); //31
    }
    $this->showMessage('La conciliaci&oacute;n fue registrada satisfactoriamente.'); //32
} //33
```

Anexo 7. Representación de las pruebas de Caja Blanca y Caja Negra

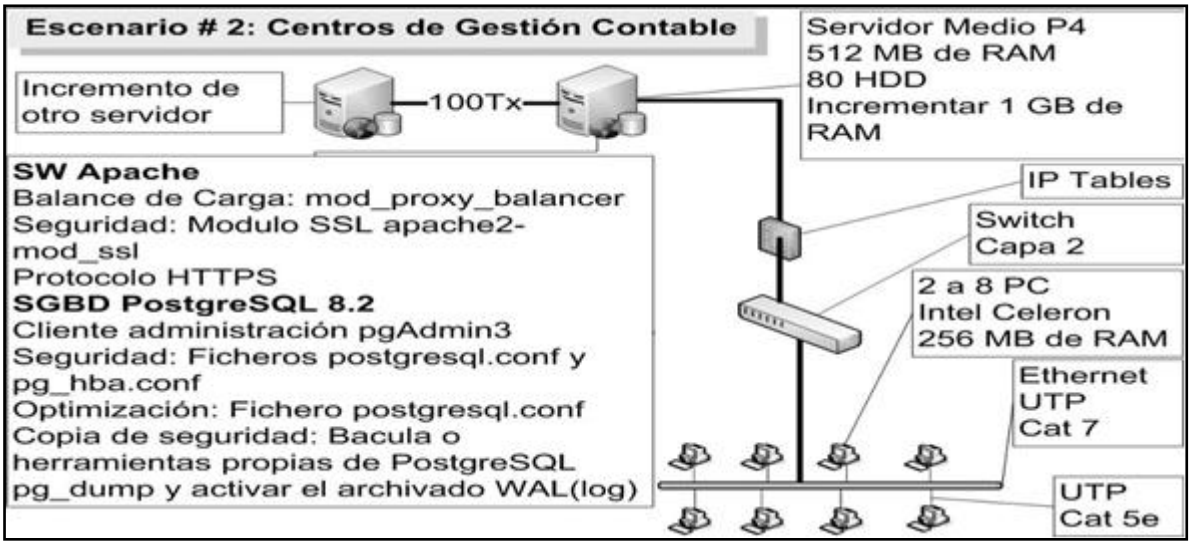


La figura representa gráficamente la filosofía de las pruebas de caja blanca y caja negra. Como se puede observar las pruebas de caja blanca necesitan conocer los detalles procedimentales del código, mientras que las de caja negra únicamente necesitan saber el objetivo o funcionalidad que el código ha de proporcionar.

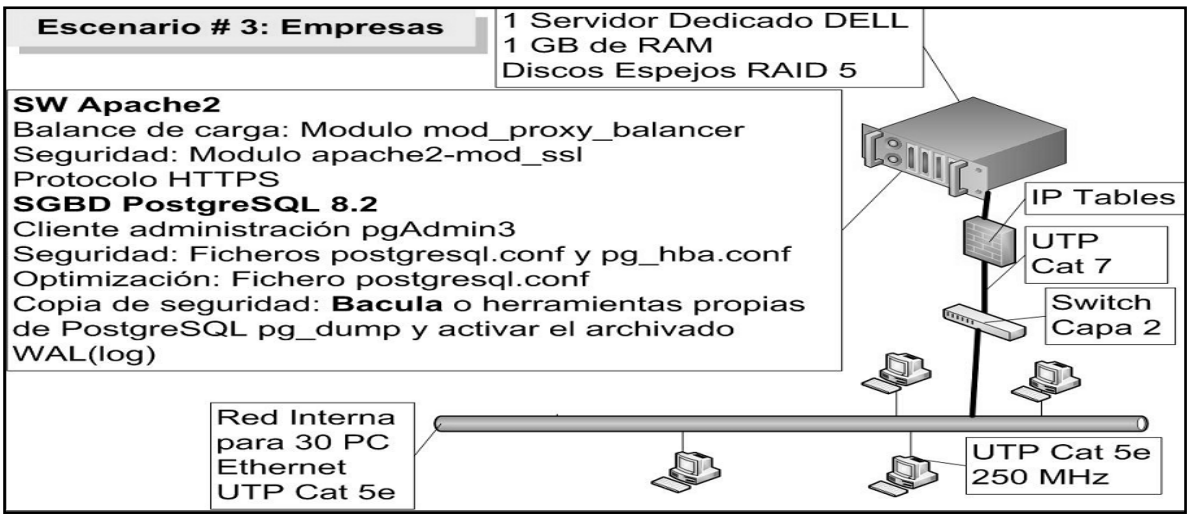
Anexo 8. Despliegue - Escenario # 1: Clientes Ligeros



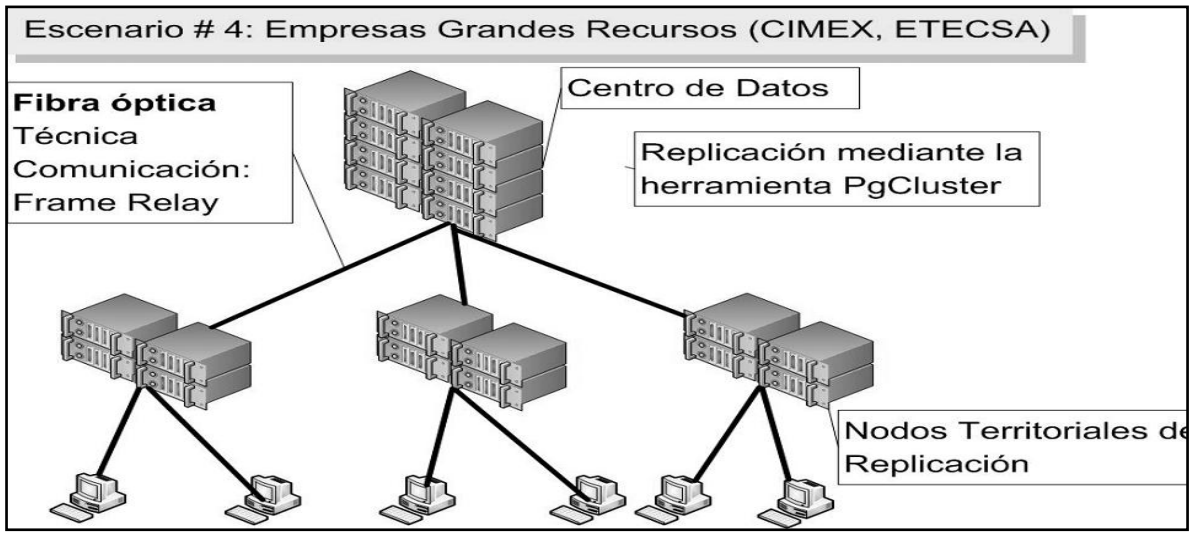
Anexo 9. Despliegue - Escenario # 2: Centros de Gestión Contable



Anexo 10. Despliegue - Escenario # 3: Empresas

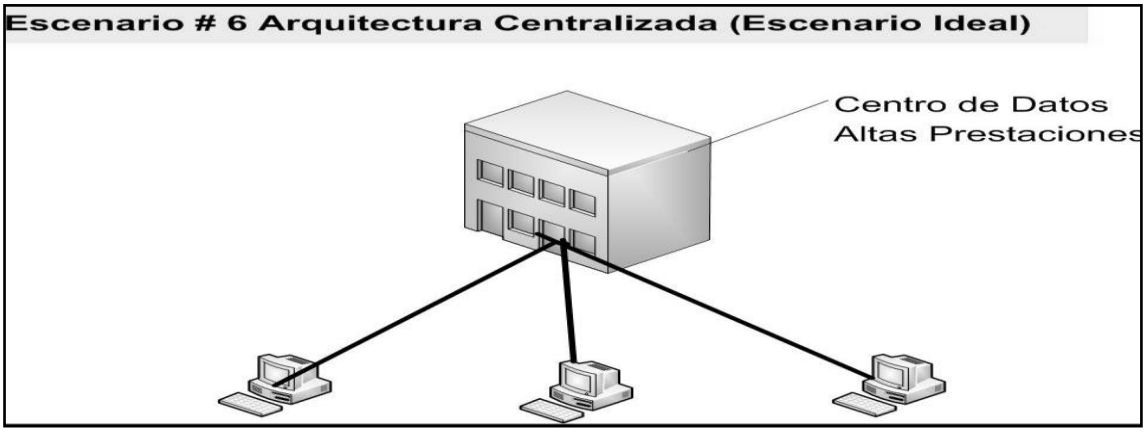


Anexo 11. Despliegue - Escenario # 4: Empresas Grandes Recursos (CIMEX, ETECSA)



Anexo 12. Despliegue - Escenario # 5: Entidad que no tienen nada

Anexo 13. Despliegue - Escenario # 6: Arquitectura Centralizada (Escenario Ideal)



Anexo 14. Instrumento de medición de la métrica Tamaño operacional de clase (TOC).

Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Tabla 8 Rango de valores de para la evaluación técnica de los atributos de calidad (Complejidad de Implementación y Reutilización) relacionados con la métrica TOC.

Clase	Cantidad de Procedimientos	Complejidad	Reutilización
cargainicialModel	21	Baja	Alta
DatConciliacionModel	1	Media	Media
DatRefderechouobligacionModel	1	Alta	Baja
DatRefanticipoModel	1	Baja	Alta
DatConciliacion	0	Baja	Alta
DatRefanticipo	0	Baja	Alta
DatRefderechouobligacion	0	Alta	Baja
NomOperacionconciliacion	0	Baja	Alta
CambioctaanticipoModel	1	Baja	Alta
CambiocuentaModel	1	Baja	Alta
condPagoModel	1	Alta	Baja
Documento	30	Media	Media
DatLiquidacion	6	Baja	Alta
NomEstadoanticipo	4	Baja	Alta
NomTipooperacion	3	Baja	Alta
DatAnticipado	15	Baja	Alta
DatCambioctaanticipo	2	Baja	Alta
DatCambiocuenta	8	Alta	Baja
DatCambiomoneda	15	Baja	Alta
DatCancelacionanticipo	3	Media	Media

DatCancelacioncontrato	3	Baja	Alta
DatCancelaciondcop	5	Baja	Alta
DatContrato	16	Baja	Alta
DatDiferencialiquidacion	1	Baja	Alta
DatDoccambiocuenta	4	Baja	Alta
DatDoccancelacion	6	Baja	Alta
DatDocumentocp	17	Baja	Alta
DatInstrumento	12	Baja	Alta
DatLetra	4	Baja	Alta
CierreContableModel	3	Baja	Alta
condicionpagoModel	2	Baja	Alta
MotivoCancelacionModel	2	Baja	Alta
parrafoModel	4	Baja	Alta
DatCondicionpago	14	Baja	Alta
NomMotivocancelacion	6	Media	Media
NomParrafofiscal	12	Baja	Alta
ContabilizarCambioCuenta	3	Media	Media
ContabilizarCobroPagoAnticipado	2	Baja	Alta
ContabilizarCambioMoneda	2	Baja	Alta
ContabilizarCobroPagoAnticipado	2	Media	Media
ContabilizarLiquidacionCobroPagoDerechoObligacion	2	Alta	Baja
CancelacionModel	1	Alta	Baja
ContabilizarCancelacionCobroPagoAnticipado	2	Media	Media
ContabilizarCancelacionDerechoObligacion	2	Baja	Alta
DerechouObligacionModel	2	Baja	Alta
ContabilizarDerechoObligacion	1	Baja	Alta
DatClienteModel	1	Baja	Alta
DatDerechouobligacionModel	6	Baja	Alta
DatOperacioncpModel	4	Baja	Alta
derechosdecobroModel	2	Baja	Alta
derechosfiscalesModel	8	Baja	Alta
DerOblig	4	Baja	Alta
obligacionesfiscalesModel	9	Alta	Baja
obligacionpagoModel	2	Baja	Alta
DatRegistroanexoopModel	0	Baja	Alta
DatCliente	8	Baja	Alta
DatDerechouobligacion	53	Baja	Alta
DatOperacioncp	16	Baja	Alta
DatRegistroanexoop	7	Baja	Alta
NomEstadodcop	4	Baja	Alta
NomEstadodocumento	4	Baja	Alta
NomTipo	4	Baja	Alta
NomTipoDerecho	4	Baja	Alta
DatFiscal	5	Baja	Alta
SubmayorclienteModel	3	Baja	Alta

Tabla 9: Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad de Implementación y Reutilización)

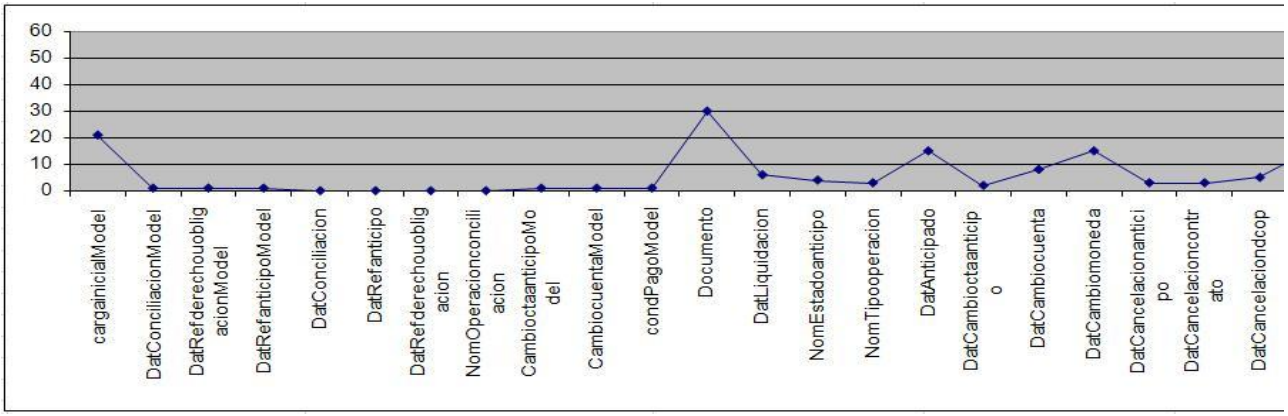


Figura 14: Gráfica de los resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad de Implementación y Reutilización), parte 1

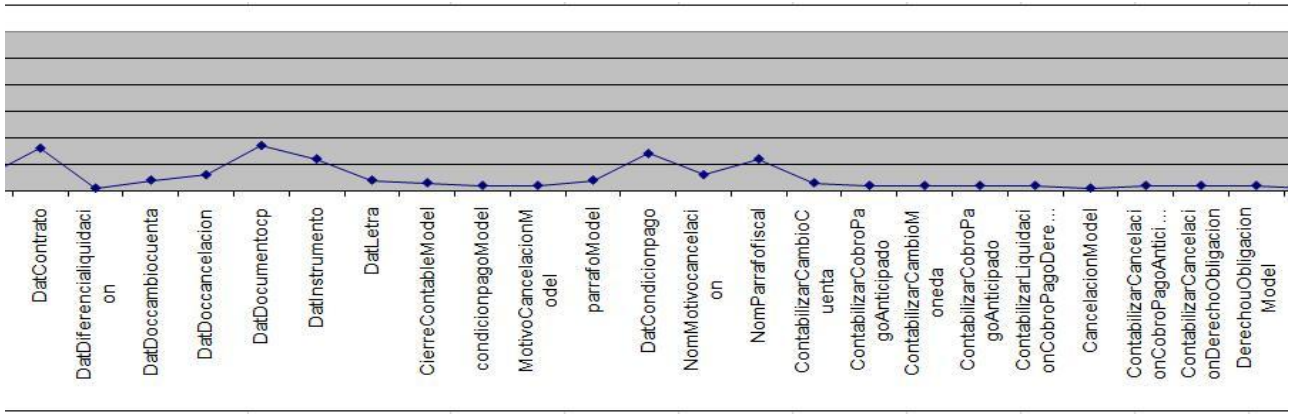


Figura 15: Gráfica de los resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad de Implementación y Reutilización), parte 2

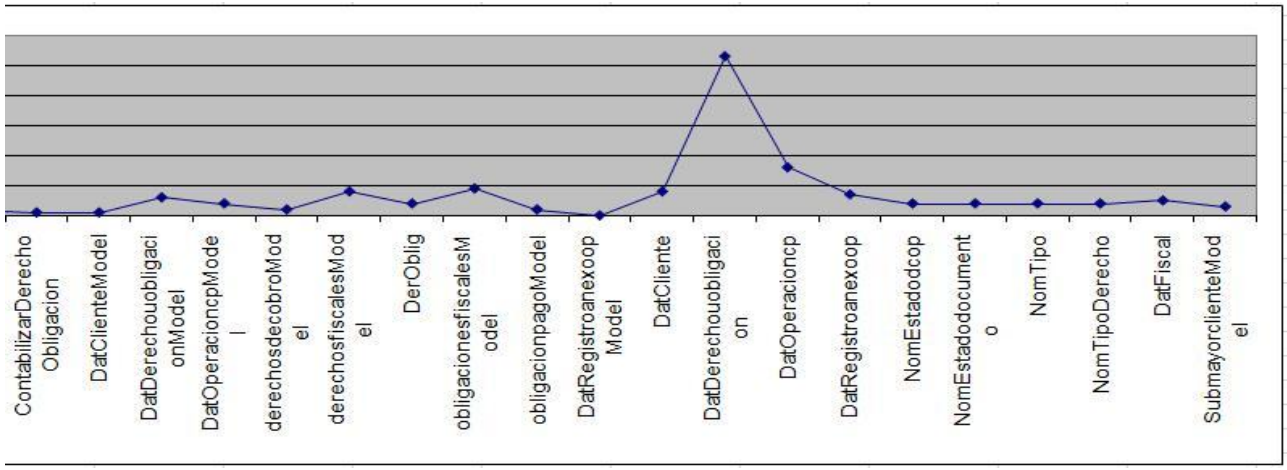


Figura 16: Gráfica de los resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad y Reutilización), parte 3

Anexo 15. Instrumento de medición de la métrica Relaciones entre clases (RC).

	Categoria	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoria	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoria	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoria	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Tabla 10: Rango de valores de para la evaluación técnica de los atributos de calidad (Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas) relacionados con la métrica RC.

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mtto.	Reutilización	Cantidad de Pruebas
cargainicialModel	7	Alto	Alta	Baja	Alta
DatConciliacionModel	1	Bajo	Baja	Alta	Baja
DatRefderechouobligacionModel	1	Bajo	Baja	Alta	Baja
DatRefanticipoModel	1	Bajo	Baja	Alta	Baja
DatConciliacion	0	Ninguno	Baja	Alta	Baja
DatRefanticipo	0	Ninguno	Baja	Alta	Baja
DatRefderechouobligacion	0	Ninguno	Baja	Alta	Baja
NomOperacionconciliacion	0	Ninguno	Baja	Alta	Baja
CambioctaanticipoModel	1	Bajo	Baja	Alta	Baja
CambiocuentaModel	1	Bajo	Baja	Alta	Baja
condPagoModel	1	Bajo	Baja	Alta	Baja
Documento	9	Alto	Alta	Baja	Alta
DatLiquidacion	2	Medio	Baja	Alta	Baja
NomEstadoanticipo	1	Bajo	Baja	Alta	Baja
NomTipooperacion	1	Bajo	Baja	Alta	Baja
DatAnticipado	2	Medio	Baja	Alta	Baja
DatCambioctaanticipo	1	Bajo	Baja	Alta	Baja
DatCambiocuenta	1	Bajo	Baja	Alta	Baja
DatCambiomoneda	4	Alto	Media	Media	Media
DatCancelacionanticipo	1	Bajo	Baja	Alta	Baja
DatCancelacioncontrato	1	Bajo	Baja	Alta	Baja
DatCancelaciondcop	1	Bajo	Baja	Alta	Baja
DatContrato	4	Alto	Media	Media	Media

DatDiferencialiquidacion	1	Bajo	Baja	Alta	Baja
DatDoccambiocuenta	1	Bajo	Baja	Alta	Baja
DatDoccancelacion	2	Medio	Baja	Alta	Baja
DatDocumentocp	3	Alto	Media	Media	Media
DatInstrumento	2	Medio	Baja	Alta	Baja
DatLetra	1	Bajo	Baja	Alta	Baja
CierreContableModel	1	Bajo	Baja	Alta	Baja
condicionpagoModel	2	Medio	Baja	Alta	Baja
MotivoCancelacionModel	1	Bajo	Baja	Alta	Baja
parrafoModel	1	Bajo	Baja	Alta	Baja
DatCondicionpago	4	Alto	Media	Media	Media
NomMotivocancelacion	2	Medio	Baja	Alta	Baja
NomParrafofiscal	4	Alto	Media	Media	Media
ContabilizarCambioCuenta	1	Bajo	Baja	Alta	Baja
ContabilizarCobroPagoAnticipado	1	Bajo	Baja	Alta	Baja
ContabilizarCambioMoneda	1	Bajo	Baja	Alta	Baja
ContabilizarCobroPagoAnticipado	1	Bajo	Baja	Alta	Baja
ContabilizarLiquidacionCobroPagoDerechoObligacion	1	Bajo	Baja	Alta	Baja
CancelacionModel	1	Bajo	Baja	Alta	Baja
ContabilizarCancelacionCobroPagoAnticipado	2	Medio	Baja	Alta	Baja
ContabilizarCancelacionDerechoObligacion	2	Medio	Baja	Alta	Baja
DerechouObligacionModel	2	Medio	Baja	Alta	Baja
ContabilizarDerechoObligacion	1	Bajo	Baja	Alta	Baja
DatClienteModel	1	Bajo	Baja	Alta	Baja
DatDerechouobligacionModel	2	Medio	Baja	Alta	Baja
DatOperacioncpModel	1	Bajo	Baja	Alta	Baja
derechosdecobroModel	2	Medio	Baja	Alta	Baja
derechosfiscalesModel	2	Medio	Baja	Alta	Baja
DerOblig	1	Bajo	Baja	Alta	Baja
obligacionesfiscalesModel	2	Medio	Baja	Alta	Baja
obligacionpagoModel	2	Medio	Baja	Alta	Baja
DatRegistroanexoopModel	0	Ninguno	Baja	Alta	Baja
DatCliente	3	Alto	Media	Media	Media
DatDerechouobligacion	8	Alto	Alta	Baja	Alta
DatOperacioncp	2	Medio	Baja	Alta	Baja
DatRegistroanexoop	3	Alto	Media	Media	Media
NomEstadodcop	1	Bajo	Baja	Alta	Baja
NomEstadodocumento	2	Medio	Baja	Alta	Baja
NomTipo	1	Bajo	Baja	Alta	Baja
NomTipoDerecho	3	Alto	Media	Media	Media
DatFiscal	2	Medio	Baja	Alta	Baja
SubmayorclienteModel	1	Bajo	Baja	Alta	Baja

Tabla 11. Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad (Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas)

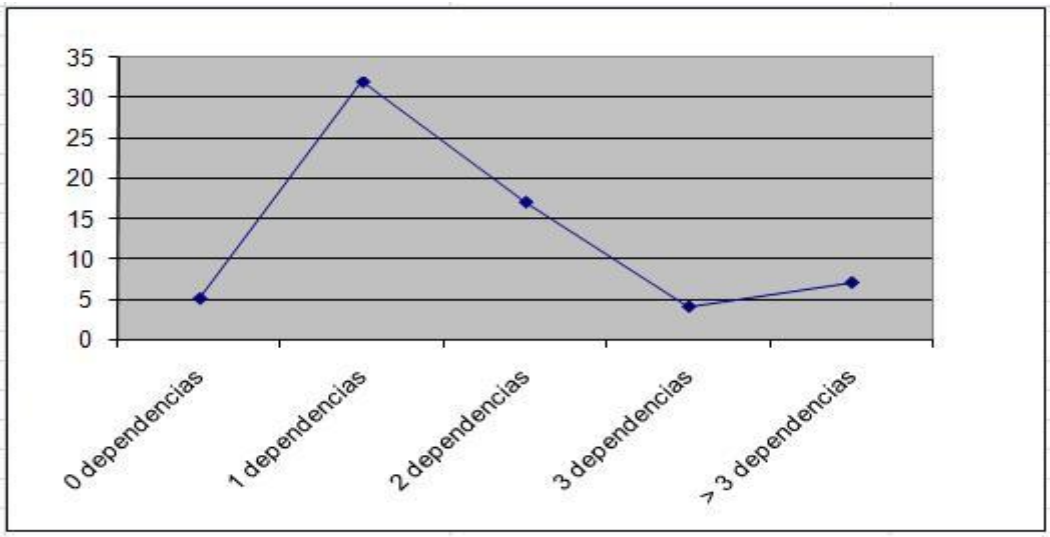


Figura 17: Gráfica de los resultados de la evaluación de la métrica RC agrupados por la tendencia de los valores.

Anexo 16. Diagrama de componentes de módulo Cobros y Pagos.