Universidad de las Ciencias Informáticas

Facultad 3

Título:

Solución Informática de los procesos solicitud y aprobación del módulo Gestión de Créditos del Sistema Integral de Gestión CEDRUX

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

Autor: Dasiel Alejandro Canino Toledo.

Tutor: Ing. Silvia María Llach Leiva.

Co-tutor: Ing. Jesús Vilches Pupo

La Habana, junio de 2013

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año_____.

_____

Dasiel Alejandro Canino Toledo

Firma del Autor

_____

Ing. Silvia María Llarch Leyva

Firma del Tutor

_____

Ing. Jesús Vilches Pupo

Firma del Co-tutor

"Seamos realistas y hagamos lo imposible."

Ernesto "Che" Guevara.

DATOS DEL CONTACTO

Síntesis del Tutor:

- **Ing. Silvia María Llarch Leyva:** Ingeniera en Ciencias Informáticas. Graduada en la Universidad de las Ciencias Informáticas en el 2008, con categoría de instructor, se desempeña como profesora en la Facultad 3 y como jefa de Gestión de Créditos en el proyecto Finanzas en el centro CEIGE.

  Correo: smllarch@uci.cu

Síntesis del Co-Tutor:

- **Ing. Jesús Vilches Pupo:** Ingeniero en Ciencias Informáticas. Graduado en la Universidad de las Ciencias Informáticas en el año 2012. Actualmente se desempeña como desarrollador del proyecto Gestión de Créditos de la línea Finanzas del ERP, CEIGE, Facultad 3.

  Correo: jvilches@uci.cu

AGRADECIMIENTOS

A mi madre Deisys, que dios la tenga en la gloria, por su apoyo y confianza en todo momento, por cada segundo que paso a mi lado dándome cariño, amor, dedicación, por ser la inspiración de mi vida, por haberme dado los mejores consejos del mundo ayudándome a ser una mejor persona cada día, por quererme con tanta intensidad y darme todo lo que tuvo a su alcance, a quien no sólo agradezco sus consejos y el apoyo constante, sino además, el privilegio de ser su hijo y aunque no esté presente sé que siempre estuvo muy orgullosa de mi, muchas gracias, sin ti nada de eso fuera posible.

A mi abuela Nena quien siempre creyó en mí, por todo el amor y cariño brindado, por sus innumerables consejos y por estar siempre presente aun cuando nada parece tener sentido.

A mi tío Abel, por ser un ejemplo a seguir, por todos sus consejos, por haberme enseñado que la vida es sacrificio y esfuerzo, y sobre todo por ser como un padre para mí.

A mi papa, al cual quiero con todas las fuerzas de mi corazón, por ser tan bueno y especial conmigo.

A toda mi Familia que siempre me han apoyado, me han educado, me han enseñado y me han dado la confianza suficiente para ser quien soy.

A Silvia por ser una tutora ejemplar, por todo su apoyo y por estar ahí siempre cuando la necesité, sin tu ayuda no fuera posible la realización de este trabajo.

A mis amigos por estar conmigo en todo este tiempo y compartir alegrías y tristezas.

Por último quiero agradecer especialmente a la Revolución por darme la oportunidad de estudiar en la UCI que ha resultado tan positiva para mi formación personal y profesional.

A todos los que de una forma u otra han contribuido a la realización de este trabajo de diploma, de corazón, muchas gracias.

DEDICATORIA

El presente trabajo se lo dedico a mi abuela y en especial a mi mamá por ser mi guía, mi ejemplo a seguir, la persona que siempre estuvo 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, 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 hiso con todo el amor del mundo para formarme como un ser integral y de las cuales me siento extremadamente orgulloso;

Madre, es a Ud. A quien le dedico esta tesis y el título que está por llegar, aunque ya no estés presente sé que este siempre fue tu sueño.

RESUMEN

Las diferentes organizaciones económicas se ven en la necesidad de transmitir información de alta calidad que esté disponible en el momento oportuno y agilizar los procesos económicos basándose en las tecnologías y la información para mejorar la toma de decisiones. En Cuba, muchas empresas optan por el uso de sistemas de gestión empresarial; sin embargo, estos sistemas no responden totalmente a las funcionalidades requeridas por las entidades para el correcto control de sus procesos empresariales. Dentro de estos procesos se encuentra la Gestión de Créditos, que garantiza la gestión y seguimiento de créditos empresariales, ayudando a mantener el control de las operaciones relacionadas con la emisión y recepción de créditos bancarios.

El presente trabajo tiene como objetivo garantizar la funcionalidad del módulo Gestión de Créditos del subsistema Finanzas del Sistema Integral de Gestión CEDRUX, partiendo del análisis de los artefactos y requerimientos establecidos para los procesos solicitud y aprobación, realizando un estudio de las tendencias modernas y buenas prácticas en el desarrollo de aplicaciones web. Durante la investigación se describen las etapas del diseño para el módulo siguiendo el Modelo de Desarrollo Orientado a Componentes establecido por el equipo de arquitectura del centro y se definen las herramientas a emplear para implementar los componentes.

El sistema garantizará el buen funcionamiento de los procesos crediticios en las entidades cubanas, cumpliendo con las legislaciones financieras actuales y constituyendo un aporte decisivo a la informatización y a la economía del país.

ÍNDICE

DECLARACIÓN DE AUTORÍA 1
DATOS DEL CONTACTO 1
AGRADECIMIENTOS 1
DEDICATORIA 1
RESUMEN 1
INTRODUCCIÓN 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN 7
1.1. Introducción 7
1.2. Sistema Integral de Gestión CEDRUX 7
1.3. Módulo de Gestión de Créditos 7
1.3.1. Ventajas del Módulo de Gestión de Créditos. 8
1.4. Análisis de sistemas financieros extranjeros. 8
1.4.1. ERP Kepler 8
1.4.2. SAP R/3. 9
1.4.3. Sage ERP X3. 9
1.5. Análisis de sistemas financieros nacionales. 10
1.5.1. SISCONT- 5. 10
1.5.2. VERSAT-Sarasola. 10
1.6. Valoración de los sistemas estudiados. 11
1.7. Arquitectura basada en componentes. 11
1.8. Metodología y Modelo de Desarrollo. 12
1.9. Ambiente de desarrollo 13

1.9.1. Lenguaje Unificado de Modelado (UML 6.4) 13
1.9.2. Herramienta Case 13
1.9.3. Lenguajes de programación 14
1.9.3.1. Cliente 14
1.9.3.2. Servidor 15
1.9.4. Framework 15
1.9.5. Herramientas y tecnologías de desarrollo 17
1.9.5.1. Control de versiones 17
1.9.5.2. IDE 17
1.9.5.3. Servidores de aplicaciones 18
1.9.5.4. Navegadores web 18
1.9.6. Gestor de Base de Datos 19
1.10. Patrón de Arquitectura 19
1.11. Patrones de diseño 20
Patrones GRASP: Para la asignación de responsabilidad 20
1.12. Pruebas de SW 23
1.12.1. Pruebas de Caja Negra 23
1.12.2. Pruebas de Caja Blanca 24
1.13. Conclusiones. 24
CAPÍTULO 2: PROPUESTA DE SOLUCIÓN 26
2.1. Introducción. 26
2.2. Valoración del análisis. 26
2.3. Procesos de Solicitud y Aprobación. 26

2.4. Diseño del sistema 29
2.4.1. Patrones de Diseño Utilizados 29
2.4.2. Diagramas del diseño 31
2.4.3. Diagrama de paquetes 31
2.4.4. Diagramas de clases del diseño con estereotipos web. 33
2.4.5. Diagrama de secuencia 34
2.4.6. Diagrama de clases persistentes. 35
2.4.7. Diagrama Entidad-Relación (DER). 36
2.5. Métricas para la evaluación del diseño 39
2.5.1. La métrica TOC 39
2.5.2. La métrica RC 45
2.6. Conclusiones 52
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA 53
3.1. Introducción 53
3.2. Estándar de codificación 53
3.2.1. PascalCasing 53
3.2.2. Nomenclatura de clases según su tipo 53
3.2.3. CamelCasing 54
3.3. Diagrama de despliegue 54
3.4. Pruebas del software 56
3.4.1. Pruebas funcionales 56
Descripción de variable 57
3.4.2. Pruebas de Caja Blanca 59

3.5. Integración entre componente 64
3.6. Conclusiones 65
CONCLUSIONES 66
RECOMENDACIONES 67
REFERENCIAS 68
ANEXO 1 70
ANEXO 2 76
ANEXO 3 80
ANEXO 4 86

INTRODUCCIÓN

El avance alcanzado por las Tecnologías de la Información y las Comunicaciones (TIC) soporta numerosos cambios que se evidencian a gran escala en el desarrollo del mundo empresarial. Las empresas dejan de ver a las TIC como un componente básico del negocio, usado solo de manera espontánea, que no produce ningún cambio en su desarrollo económico y comienzan a enfocarlo como un elemento crítico y primordial para la ejecución de sus estrategias productivas. Las diferentes organizaciones, enmarcadas en el sistema empresarial competitivo que se vive actualmente, necesitan contar con información confiable y oportuna para el cumplimiento de sus principales objetivos, por lo que demandan la creación de sistemas que les permitan controlar y gestionar la información que manejan con el fin de tomar mejores decisiones.

Debido a esto surgen los Sistemas de Planificación de Recursos Empresariales (en inglés ERP, Enterprise Resource Planning) que integran y automatizan muchas de las prácticas de negocio asociadas a los aspectos operativos y productivos de una empresa, con el objetivo de facilitar la gestión de todos sus recursos, a través de la integración de la información de los distintos departamentos y áreas funcionales, permitiendo que el acceso a esta sea confiable.

Esto conlleva a que Cuba vea como aspecto primordial en su desarrollo económico, la necesidad de obtener una herramienta que se ajuste al actual perfeccionamiento empresarial, integre todos los procesos de negocio y se adapte a su economía.

Dentro de la estructura productiva de la Universidad de las Ciencias Informáticas (UCI) se encuentra el Centro de Informatización de Gestión de Entidades (CEIGE), el cual dirige proyectos con el objetivo de solventar las necesidades económicas cubanas, a través del desarrollo de sistemas que brindan numerosas soluciones empresariales. El centro CEIGE en coordinación con el Ministerio de Finanzas y Precios junto a las Entidades de Desarrollo de Software (DESOFT) y la Empresa de Información y Conocimiento del Ministerio del Azúcar Casa del Software de Villa Clara (UEB TEICO), ha venido desarrollando un ERP cubano con el objetivo de modelar y automatizar la mayoría de los procesos de negocio en las entidades económicas del país. Con este Sistema Integral de Gestión CEDRUX, se busca mejorar los resultados económicos y lograr una mejor planificación de los recursos materiales y financieros de acuerdo a las normas y legislaciones actuales, significando un paso de avance importante en el desarrollo económico demandado por el sistema empresarial cubano. CEDRUX cuenta con los

subsistemas Finanzas, Contabilidad, Capital Humano, Logística y Planificación, que permiten integrar la información que manejan los diferentes departamentos de las empresas.

Como parte de la solución del subsistema Finanzas y en respuesta a la necesidad de gestionar y controlar la actividad crediticia en las diferentes organizaciones económicas del país, se comenzó la creación en el Centro de Desarrollo Territorial de Holguín de un módulo de Gestión de Créditos, el cual posteriormente fue trasladado a la UCI para culminar su implementación y así poder ser integrado a CEDRUX. El módulo de Gestión de Créditos permite a una entidad gestionar el proceso general y los pasos necesarios de un préstamo; desde el momento que se solicita hasta que termina de amortizarse.

El equipo de trabajo en conjunto con el jefe del proyecto realizó un estudio detallado de cada requisito asociado a los procesos solicitud y aprobación del módulo Gestión de Créditos, se tomaron sus respectivos Casos de Prueba, y se le hicieron pruebas al sistema arrojando como resultado la existencia de inconformidades en la implementación de los mismos que afectan la continuidad del desarrollo del módulo debido a que al existir problemas en alguno de los procesos se interrumpe el flujo lógico que estos deben seguir. Además se identificaron inconsistencias en la integración que se debe llevar a cabo con los módulos del subsistema finanzas y se determinaron problemas en la implementación de los servicios que garantizan la integración del módulo con los restantes subsistemas que comprende CEDRUX. Debido a lo descrito anteriormente se concluyó que la solución no es funcional. Algunas de las deficiencias encontradas se describen a continuación:

- No se obtienen los datos de la cuenta bancaria, lo que provoca que no se pueda generar una oferta de crédito.
- No se pueden realizar las operaciones de cobro o pago por comisión, mora e intereses.
- Los contratos no se pueden adicionar porque no se encuentran las solicitudes que los generaron.
- No se pueden generar los derechos de cobros u obligaciones de pago debido a que no se encuentran las cuentas y operaciones contables definidas para la entidad.
- Los servicios de notificaciones implementados en el módulo no funcionan.

- No se muestran los contratos definidos en el sistema cuando se va a hacer una solicitud de restructuración o renegociación de los mismos.

- No se pueden cancelar los contratos definidos en el sistema.

- Los comprobantes que se crean al generar un derecho de cobro u obligación de pago no se pueden contabilizar.

Atendiendo a la situación problemática descrita con anterioridad se identificó el siguiente **problema a resolver:** ¿Cómo garantizar la funcionalidad del módulo Gestión de Crédito y la integración del mismo al Sistema Integral de Gestión CEDRUX?

A partir del problema definido, el **objeto de estudio** estará enfocado a los procesos de la Gestión de Créditos de los Sistemas Financieros Contables y el **campo de acción** se centra en la solución informática de los procesos solicitud y aprobación del módulo Gestión de Créditos del Sistema Integral de Gestión CEDRUX.

El **objetivo general** del trabajo se basa en realizar el diseño e implementación de los procesos solicitud y aprobación del módulo Gestión de Créditos que permitan garantizar la funcionalidad del mismo y su integración con el Sistema Integral de Gestión CEDRUX.

Para dar cumplimiento al objetivo propuesto, se proponen los siguientes **objetivos específicos:**

- **OE1.** Elaborar el marco teórico de la investigación.
- **OE2.** Realizar el diseño de la propuesta de solución.
- **OE3.** Realizar la implementación de la propuesta de solución.
- **OE4.** Validar la propuesta de solución.

La **propuesta de solución** abarcará el diseño e implementación de funcionalidades que garantizarán el estado funcional del módulo de Gestión de Créditos, así como la integración del mismo con el Sistema Integral de Gestión CEDRUX partiendo de los artefactos obtenidos durante el análisis.

Posibles resultados del trabajo se centran en la obtención de un módulo de Gestión de Créditos funcional e integrado al Sistema Integral de Gestión CEDRUX, que incluye la obtención de artefactos tales como: diagrama de clases del diseño, descripción de las clases del diseño, modelo de datos, diagrama de

despliegue, descripción de funcionalidades implementadas, validación mediante el uso de métricas y pruebas de software.

Las **tareas de la investigación** para llevar a cabo los objetivos específicos son:

OE1
➢ **Elaboración del marco conceptual de la investigación.** ➢ **Estudio de los procesos relacionados con el módulo de Gestión de Créditos.** ➢ **Realización de análisis valorativo.** ➢ **Estudio de patrones de diseño.** ➢ **Estudio de métricas para validar el diseño.**
OE2
➢ **Definición de mecanismos de diseño a utilizar.** ➢ **Realización de diagramas de clases del diseño con la aplicación de patrones.** ➢ **Descripción de las clases del diseño.** ➢ **Realización del modelo de datos.**
OE3
➢ **Definición de estándares de codificación.** ➢ **Realización del diagrama de despliegue teniendo en cuenta los aspectos**

técnicos de la solución. ➢ **Realización de descripción por funcionalidades.**
OE4
➢ **Validación del diseño mediante la aplicación de métricas.** ➢ **Validación de funcionalidades mediante la aplicación de pruebas de Caja Blanca y Caja Negra.**

Tabla #1: Tareas de la investigación

Los **métodos teóricos** que sustentan la investigación son:

Histórico-Lógico: Es utilizado con el objetivo de realizar un estudio del estado del arte del tema a investigar. De esta manera se pudo conocer acerca de la existencia y características de sistemas que contemplan la Gestión de Créditos desarrollados anteriormente.

Analítico-Sintético: Este método permitió analizar y comprender los procesos de negocio del módulo Gestión de Créditos. Resultó importante a la hora de fundamentar los elementos más significativos de la investigación determinando los aspectos esenciales y el arribo a conclusiones prácticas y teóricas.

A continuación se muestra una breve descripción de los 3 capítulos por los que está compuesto el presente trabajo de diploma:

Capítulo 1. Se plantean los principales aspectos relacionados con los sistemas informáticos vinculados al campo de acción. Se fundamenta la utilización de un conjunto de técnicas y herramientas usadas para la modelación y construcción del módulo.

Capítulo 2. Se abordan los requerimientos que presentan problemas en los procesos Solicitud y Aprobación, se realiza el diseño de la propuesta de solución, presentando los diagramas pertinentes que describan dicho diseño y se valida el diseño realizado mediante la aplicación de métricas para determinar su nivel de calidad.

Capítulo 3: Se realiza la implementación teniendo en cuenta los estándares de codificación descritos, se valida el componente a partir del diseño realizado aplicando las pruebas de Caja Blanca y Caja Negra y se describen los puntos a tener en cuenta para el despliegue de la aplicación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

1.1. Introducción

En el presente capítulo se describen los elementos principales que fundamentan el contenido de la investigación. Se realiza un análisis de los sistemas ERP existentes tanto en Cuba como en el resto del mundo haciendo énfasis en como realizan la Gestión de Créditos y se detallan aspectos importantes de las distintas tecnologías que serán empleadas durante el desarrollo del módulo.

1.2. Sistema Integral de Gestión CEDRUX

El Sistema Integral de Gestión CEDRUX es un paquete de soluciones integrales de gestión para las entidades presupuestadas y empresariales basada en los principios de independencia tecnológica y con funcionalidades generales de los procesos y las particularidades de la economía cubana. Dentro de las principales características de CEDRUX se encuentra su independencia tecnológica. Funciona sobre plataforma web y puede mantener interoperabilidad con otros sistemas. Es multimoneda, multientidades y como característica particular de Cuba permite hacer las operaciones sobre la doble moneda (peso cubano y peso convertible). Mantiene el control de fechas, es transaccional, con integridad funcional y posibilita el tratamiento estadístico del procesamiento de la información. Este cuenta con varios subsistemas como Contabilidad General, Costos y Procesos, Capital Humano, Planificación, Logística y Finanzas. Como parte de la solución del subsistema de Finanzas se encuentra el módulo de Gestión de Créditos, el cual no está en funcionamiento debido a las deficiencias detectadas para los procesos solicitud y aprobación, las cuales dificultan el paso al proceso de otorgamiento imposibilitando así la completa concesión del crédito. Dicho modulo se describe en el siguiente acápite.

1.3. Módulo de Gestión de Créditos

El módulo de Gestión de Créditos garantiza la gestión y seguimiento de créditos empresariales, ayudando a mantener el control de las operaciones relacionadas con la emisión y recepción de créditos bancarios, a partir de la definición de los tipos de créditos que la entidad necesita u otorga a terceros, calcula y contabiliza diariamente o en un período seleccionado los intereses bancarios, prorrogando, pasando a vencidos y cancelando los créditos definidos en el sistema y además permite emitir una variedad de reportes sobre el estado de los saldos en cada tipo de crédito.

1.3.1. Ventajas del Módulo de Gestión de Créditos

Las ventajas presentes en el módulo se enfocan según las prestaciones que brindan, las cuales son:

- Facilita la actualización del estado del préstamo en todo momento.
- Permite configurar el módulo según las características de la entidad.
- Crea solicitudes y ofertas de créditos, de diferentes tipos, previamente definidos.
- Establece contratos de créditos bancarios empresariales.
- Da seguimiento a los otorgamientos y amortizaciones del crédito asignado.
- Genera órdenes automáticas de cobros o pagos en dependencia del estado del crédito.
- Genera diversas recuperaciones o salidas del sistema según diferentes criterios.

Con el uso de este módulo se abre una puerta al perfeccionamiento empresarial de Cuba teniendo en cuenta que las propias empresas podrán emitir diferentes tipos de créditos u otorgar a terceros según su necesidad, manteniendo el control y supervisión de todo el proceso mediante la contabilización diaria y la emisión de reportes.

1.4. Análisis de sistemas financieros extranjeros

1.4.1. ERP Kepler

Es una solución de negocios integral, flexible, superior a los ERP para medianas empresas existentes, de fácil instalación y muy corto período de implementación; Kepler es un sistema integrado, sencillo e intuitivo, con más de 30 años de éxitos en el mercado del software empresarial. Cuenta con módulos de Logística, Almacenes, Producción y Finanzas. Todos estos módulos están perfectamente integrados. Este software ERP es flexible, sencillo y perfectamente adaptable a la dinámica de su negocio con un instrumento intuitivo cuyo objetivo es la satisfacción del usuario final. Su módulo de Gestión de Créditos contempla todas las modalidades de crédito del sistema financiero peruano. El sistema es totalmente parametrizable. Se adapta a las condiciones de operación de la Institución Financiera para definir los productos de créditos en forma particular (Tasa de interés fija o variable, cualquier período de pago, multimoneda). Los créditos se pueden administrar a través de líneas de crédito revolventes o no. (1)

1.4.2. SAP R/3

Ofrece un potente entorno de trabajo para la Gestión de Créditos. Al utilizar información integrada y actual de Gestión Financiera (FI) y de Gestión Comercial (SD), el sistema permite implementar una política de créditos que minimiza de manera efectiva el riesgo de créditos, resuelve los documentos bloqueados por motivo de crédito lo más rápidamente posible, y acelera la gestión de pedidos. La Gestión de Créditos incluye las siguientes características:

Dependiendo de las necesidades de la Gestión de Créditos, se pueden especificar verificaciones de crédito automáticas basadas en un conjunto de criterios. También se puede especificar en qué puntos críticos del ciclo comercial se efectuarán estas verificaciones. La Gestión de Créditos permite determinar el riesgo de crédito especificando un límite de crédito para un cliente. De esta forma se puede estimar la salud financiera de un cliente o grupo de clientes, identificar señales de aviso con prontitud y desarrollar la búsqueda de decisiones en cuanto a créditos. Esto resulta muy útil si los clientes se encuentran en industrias o empresas financieramente inestables o si realizan operaciones comerciales con países que son políticamente inestables o utilizan una política de tipo de cambio restrictiva. La integración en SAP se logra a través de la puesta en común de la información de cada uno de los módulos y por la alimentación de una Base de Datos común. Por lo tanto, se debe tener en cuenta que toda la información que se introduce en SAP repercutirá, al momento, a todos los demás usuarios con acceso a la misma. Este hecho implica que la información siempre debe estar actualizada, debe ser completa y debe ser correcta. (2)

1.4.3. Sage ERP X3

Integra toda la información y los procesos de la empresa en un único sistema de software y Base de Datos, permitiendo que los usuarios tengan una visión global de su actividad en tiempo real, independientemente del lugar donde se han creado o almacenado los datos. El sistema gestiona las operaciones de Finanzas, Ventas, Inventarios, CRM, Compras y Producción de forma global y racionaliza todos los procesos de la empresa.

Sage X3 Crédito es una solución para Cooperativas con Sección de Crédito que contempla las funcionalidades de una gestión bancaria. Ha sido desarrollado con las mismas herramientas que los módulos que conforman Sage X3, aportando soluciones a las necesidades específicas del sector. Sage X3 Crédito trabaja en entornos abiertos (Windows 95, Windows 98, Windows 2000, Windows NT,

Windows XP y UNIX) y con bases de datos SQL Server y Oracle, en modo cliente / servidor, o en modo Web (Intranet / Internet). Gracias al módulo de desarrollo se puede adaptar la aplicación a las exigencias de la actividad particular de cada Cooperativa. También se puede utilizar cualquier lenguaje del mercado para realizar adaptaciones, apoyándose en las API de acceso a los objetos. Las pantallas existen tanto en modo gráfico como en modo carácter para los procesos móviles, sigue la ergonomía de Windows (lista izquierda, pestañas, botones de acción), facilita el uso del ratón, dispone de una ayuda en línea, de la posibilidad de consultar por zoom vía túneles parametrizables. (3)

1.5. Análisis de sistemas financieros nacionales

1.5.1. SISCONT- 5

Fue creado por TECNOMATICA, surgió el 30 de junio de 1980, es un software que tiene varios módulos integrados (Contabilidad, Tesorería, Caja Chica, Créditos y Cobranzas, Gestión de Negocio, Presupuestos, Informes Gerenciales, Motor: Información compartida entre sistemas) con diversas soluciones para su contabilidad y finanzas. A los módulos integrados se le pueden adicionar datos desde otras soluciones para mayor control de su negocio, para esto se cuenta con el Motor SISCONT que hace que todos compartan información agilizando todos los procesos de la entidad. El sistema presenta desventajas puesto que toda la gestión de la aplicación se hace a través de ventanas independientes haciendo más engorroso el trabajo, además de que solo trabaja sobre el sistema operativo Windows. SISCON cuenta con un módulo de Créditos y Cobranza el cual permite la gestión de cobranza de clientes con herramientas de mantenimiento de cartera, que actualiza en línea su flujo de caja y tiene una pantalla que facilita la cancelación del cliente con diferentes tipos de pago y registros automáticos de la cancelación de retenciones. (4)

1.5.2. VERSAT-Sarasola

Fue desarrollado en 1998 por TEICO Villa Clara, empresa del Ministerio del Azúcar encargada de la Informática y las Comunicaciones, siendo el primer sistema de contabilidad cubano certificado. Automatiza las actividades de planificación, control y análisis económico de cualquier tipo de entidad, abarcando la administración, contabilidad, medios de rotación, activos fijos, finanzas, cajas y costos. Está implementado con modernas tecnologías y trabaja en red con alta seguridad. Está constituido por 12 módulos entre los vuales se incluyen Configuración y Seguridad, Contabilidad General y de Gastos, Costos y Procesos,

Análisis Económico Empresarial y Control de Activos Fijos. Además, interviene Finanzas y Cajas, Planificación y Presupuestos, Control de Inventarios, de Productos Terminados, Pago de Salario, Paquete de Gestión, Contratación y Facturación. VERSAT permite trabajar con diferentes monedas (multimoneda) para revalorizar los Estados Financieros en una fecha y con una tasa de cambio determinada. Además, posibilita procesar los documentos primarios en los diferentes subsistemas en doble moneda y por tanto obtener los resultados que se deseen en cada una de las monedas utilizadas. El sistema VERSAT Sarasola fue seleccionado por el Ministerio de Finanzas y Precios (MFP) como la herramienta informática más adecuada para implantarse en varios Centros de Gestión Contables del país. Cuenta con un módulo de Gestión de Créditos Bancarios que le permite controlar las operaciones relacionadas con la emisión y recepción de créditos bancarios, sus principales características se describen a continuación.

- Permite prorrogar, pasar a vencidos y cancelar los créditos definidos en el sistema.
- Brinda la posibilidad de emitir reportes sobre el estado de los saldos.
- Define los tipos de créditos que la entidad necesita u otorga a terceros.
- Calcula y contabiliza los intereses bancarios. (5)

1.6. Valoración de los sistemas estudiados

Luego de un detallado análisis de los ERP citados con anterioridad, haciendo énfasis en sus módulos de gestión de crédito, se llegó a la conclusión de que no existe un sistema que integre toda la información de las empresas cubanas y se adapte a las necesidades propias de la economía del país. A pesar de que no son las soluciones más factibles para el sistema económico cubano muchos de estos sistemas presentan características que pueden ser adaptadas al módulo de Gestión de Créditos del Sistema Integral de gestión CEDRUX. Las características detectadas son la posibilidad del sistema SAP R/3 de determinar el riesgo en los créditos especificando un límite de crédito para un cliente o grupo de clientes, así se puede estimar su salud financiera y determinar si estos provienen de empresas financieramente inestables. El sistema VERSAT Sarasola permite emitir variedad de reportes sobre el estado de los saldos en cada tipo de crédito, las cuales son características factibles a implementar en el módulo en desarrollo.

1.7. Arquitectura basada en componentes

Por las características presentes en el dominio del negocio y las tendencias de desarrollo de otros sistemas ERP, se decide adoptar por el equipo de arquitectura del centro CEIGE para el desarrollo horizontal del sistema el estilo arquitectónico orientado a componentes. Los componentes son porciones

ejecutables de distribución independiente, con una interfaz claramente definida, preparadas para ser utilizadas por terceros, que no tienen un estado persistente y que interactúan para formar un sistema funcional. (6)

El disponer de componentes de software no es suficiente para desarrollar aplicaciones, provengan estos de un mercado global o sean desarrollados para una aplicación específica. Un aspecto crítico a la hora de construir sistemas complejos es el diseño de la estructura del sistema, y por ello el estudio de la Arquitectura de Software se ha convertido en una disciplina de especial relevancia en la ingeniería del software. (7)

Las arquitecturas de software basadas en componentes brindan el soporte para la integración de partes en sistemas mayores, facilitando la definición de una estructura de ensamblado adecuada. El empleo de esta técnica de desarrollo de software requiere por lo tanto de un cuidadoso modelado arquitectural y análisis, con el fin de asegurar reusabilidad y compatibilidad entre los componentes a interactuar. (8)

1.8. Metodología y Modelo de Desarrollo

Para el desarrollo del módulo de Gestión de Créditos se siguieron los lineamientos arquitectónicos establecidos por la dirección del proyecto que plantean la definición clara de cada una de las responsabilidades de los diferentes roles que intervienen en el desarrollo de la solución así como el uso del modelo de desarrollo de Software establecido por el CEIGE.

Este modelo de desarrollo se caracteriza por:

Centrado en la arquitectura

La arquitectura determina la línea base y los elementos de software estructurales a partir de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo.

Orientado a componentes

Las iteraciones son orientadas por el nivel de significación arquitectónica de los componentes, que constituyen abstracciones de los procesos de negocio y requisitos asociados a modelar; y establecen al componente como unidad de medición y ordenamiento de las iteraciones.

Iterativo e incremental

Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, que son integrados permitiendo la evolución incremental del producto.

Ágil y adaptable al cambio

El desarrollo de las partes formaliza las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en e l proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados. (9)

1.9. Ambiente de desarrollo

Para la elaboración del producto CEDRUX el equipo de arquitectura del centro CEIGE definió el ambiente de desarrollo a utilizar, el cual se describe a continuación.

1.9.1. Lenguaje Unificado de Modelado (UML 6.4)

Es el lenguaje de modelado de sistemas de software, respaldado por el Object Management Group (OMG). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. (10)

1.9.2. Herramienta Case

VisualParadigm 6.4

Visual Paradigm para UML, es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Principales características:

- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación.
- Soporta aplicaciones Web.
- Soporte de UML versión 2.1.
- Modelado colaborativo Subversión (control de versiones).
- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI.
- Generación de código.
- Diagramas de Procesos de Negocio. (11)

1.9.3. Lenguajes de programación

1.9.3.1. Cliente

Los lenguajes de programación del lado cliente son aquellos que pueden ser directamente interpretados por el navegador y no necesitan un pre-tratamiento, siendo el navegador quien soporta la carga de procesamiento. (12)

JavaScript 1.6

Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Características:

- Es simple.
- Maneja objetos dentro de una página Web y sobre ese objeto permite definir diferentes eventos.
- Es dinámico, responde a eventos en tiempo real. (13)

AJAX

AJAX se basa en el modelo de aplicación Web asíncrona, son aplicaciones interactivas, rápidas y veloces. Se basa en la combinación de 4 tecnologías existentes como JavaScript, XML, HTML y CSS**.** (14)

1.9.3.2. Servidor

Los lenguajes de programación del lado servidor son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. Estos se ejecutan en el servidor web, justo antes de que se envíen páginas al cliente. Estas páginas pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente. (12)

PHP 5.2

PHP es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor.

Ventajas:

- Soporte para una gran cantidad de bases de datos.
- Integración con varias bibliotecas externas (permite generar documentos de Acrobat Reader así como analizar código XML).
- Ofrece una solución simple y universal para las paginaciones dinámicas de la web de fácil programación. (15)

1.9.4. Framework

El término framework se refiere a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Entre los objetivos principales que se persiguen con el uso de frameworks están: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Zend Framework 1.9.7

Se trata de un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5. Es

una implementación que usa código 100% orientado a objetos. La estructura de los componentes es única; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado.

Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend Framework conforman un potente y extensible framework de aplicaciones web al combinarse. Ofrece una abstracción de Base de Datos fácil de usar, y un componente de formularios que implementa la prestación de formularios HTML, validación y filtrado para que los desarrolladores puedan consolidar todas las operaciones usando de una manera sencilla la interfaz orientada a objetos. (16)

Doctrine 1.2.1

Es una librería para PHP que permite trabajar con un esquema de Base de Datos como si fuese un conjunto de objetos, y no de tablas y registros. Doctrine está inspirado en Hibérnate, que es uno de los Object Relational Mapper (ORM) que brinda una capa de abstracción de la Base de Datos muy completa. La característica más importante es que da la posibilidad de escribir consultas de Base de Datos en un lenguaje propio llamado Doctrine Query Language (DQL). (17)

ExtJs 2.2

Es una biblioteca de Java Script para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Permite crear aplicaciones complejas utilizando componentes predefinidos.

Usar ExtJS permite tener los siguientes beneficios:

- Existe un balance entre Cliente – Servidor. La carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- Comunicación asíncrona. En este tipo de aplicación el motor de render puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario.
- Eficiencia de la red. El tráfico de red puede disminuir al permitir que la aplicación elija qué información desea transmitir al servidor y viceversa, sin embargo la aplicación que haga uso de la pre-carga de datos puede que revierta este beneficio por el incremento del tráfico. (18)

1.9.5. Herramientas y tecnologías de desarrollo

1.9.5.1. Control de versiones

El control de versiones es 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. El principal objetivo es permitir editar de forma colaborativa y compartir información. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión.

SubversionSVN

Es un software libre desarrollado para el control de versiones. Es un sistema centralizado para compartir información que permite realizar modificaciones atómicas y gestionar archivos, directorios y sus cambios a través del tiempo, lo que facilita las tareas administrativas. Su capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Existen interfaces o programas individuales que integran a este sistema en entornos de desarrollo. En este proyecto en específico se utilizarán dos para la manipulación y actualización de versiones:

- **TortoiseSVN:** Provee integración con el explorador de Windows.
- **RapidSVN:** Provee integración con el sistema operativo Linux. (19)

1.9.5.2. IDE

El Entorno de Desarrollo Integrado (IDE) constituye un editor de código para depurar y facilitar las diferentes tareas necesarias en el desarrollo de cualquier tipo de aplicación que pueda funcionar con diferentes lenguajes de programación.

A continuación se listan los IDEs usados para PHP con el objetivo de seleccionar el entorno de desarrollo que mejor se adapte a las necesidades del proyecto.

Geany

Es un editor de texto ligero con características básicas de IDE. Está disponible para distintos sistemas operativos, como GNU/Linux, Mac OS X, BSD, Solaris, y Microsoft Windows. Es distribuido como software

libre bajo la Licencia Pública General de GNU.

Características:

- Autocompletado
- Soporte multidocumento
- Soporte de proyectos
- Coloreado de sintaxis
- Emulador de terminal incrustado. (20)

1.9.5.3. Servidores de aplicaciones

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

Apache 2.0

Apache es un software libre de código abierto que funciona sobre diversas plataformas, entre las cuales están Unix, Windows y Macintosh. Entre sus características se aprecia que este presenta mensajes de error altamente configurables y Base de Datos de autenticación y negociado de contenidos. Es utilizado comúnmente para sitios con páginas estáticas.

Ventajas:

- Modular (módulos cargados dinámicamente).
- Funciona sobre diversas plataformas.
- Gratuito.
- Amplias librerías disponibles.
- Código fuente seguro. (21)

1.9.5.4. Navegadores web

Los navegadores son aplicaciones capaces de interpretar las órdenes recibidas en forma de código HTML y convertirlas en las páginas que son el resultado de dicha orden. Permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web.

Mozilla Firefox 3.8

Es un navegador web libre desarrollado por la Corporación Mozilla. Es multiplataforma y está disponible en varias versiones de Microsoft Windows, Mac OS X y GNU/Linux. Su código fuente es software libre, publicado bajo una triple licencia GPL/LGPL/MPL. Mozilla Firefox es compatible con varios estándares web, incluyendo HTML, XML, XHTML, CSS, JavaScript, DOM e imágenes. (22)

1.9.6. Gestor de Base de Datos

Los servidores de bases de datos surgen por la necesidad que tienen las empresas de manejar grandes y complejos volúmenes de datos, al tiempo que requieren compartir la información con un conjunto de clientes de una manera segura. Ante este enfoque, un Sistema Gestor de Bases de Datos (SGBD) deberá ofrecer soluciones de forma fiable, rentable y de alto rendimiento. Los SGBD proporcionan herramientas de apoyo a la toma de decisiones proporcionando una Plataforma de Transacciones "On-Line" (OLTP) que hacen que la información esté siempre actualizada y consistente. Ofrecen además, las herramientas de administración completas que simplifican la tarea de la configuración, seguridad, creación y gestión de bases de datos; y facilitan los mecanismos de integración con otros sistemas, políticas de copias de seguridad y herramientas que permitan su programación tanto a nivel de diseño como a nivel de reglas o procedimientos que encapsulen la arquitectura de la Base de Datos. (23)

PostgreSQL 8.3

PostgreSQL está considerado como un SGBD de código abierto, destacándose por su robustez, escalabilidad y cumplimiento de los estándares SQL. Entre las características de PostgreSQL están: alta concurrencia, que evita tener que bloquear una tabla cuando se está escribiendo en ella; las copias de seguridad en línea, la replicación asíncrona, las transacciones anidadas y el optimizador de consultas. (24)

1.10. Patrón de Arquitectura

Modelo Vista Controlador (MVC): Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El estilo de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página.

- **El Modelo** es el objeto que representa los datos del programa. Maneja los datos y controla todas

sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

- **La Vista** es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.
- **El Controlador** es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo. (25)

1.11. Patrones de diseño

Los patrones de diseño brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Existen diversos patrones que en dependencia de su uso, permiten resolver un problema general de diseño, dentro de un contexto particular, constituyendo una vía para que los desarrolladores realicen buenas prácticas en la implementación.

Patrones GRASP: Para la asignación de responsabilidad

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Estos patrones son:

- **Experto**

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Problema: ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?

Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, se toman decisiones sobre la asignación de

responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y presenta la oportunidad de reutilizar los componentes en futuras aplicaciones.

- **Creador**

Solución: Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

1. B contiene los objetos A.
2. B agrega los objetos A.
3. B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado.
4. B registra las instancias de los objetos A.
5. B utiliza específicamente los objetos A.

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilización.

- **Bajo acoplamiento**

Solución: Asignar una responsabilidad para mantener bajo acoplamiento**.**

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Una clase con bajo (o débil) acoplamiento no depende de muchas otras.

- **Alta cohesión**

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta**.**

Problema ¿Cómo mantener la complejidad dentro de límites manejables?

En la perspectiva del desafío orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

- **Controlador**

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

1. El "sistema" global (controlador de fachada).
2. La empresa u organización global (controlador de fachada).
3. Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
4. Un manejador artificial de todos los eventos del sistema de un Caso de Uso, generalmente denominados "Manejador<NombreCasodeUso>"(controlador de casos de uso).
5. Utilice la misma clase de controlador con todos los eventos del sistema en el mismo caso de uso.

Problema: ¿Quién debería encargarse de atender un evento del sistema?

Patrones GOF

Los patrones GOF, se clasifican en 3 categorías: creacionales, estructurales y de comportamiento.

Patrones estructurales: Son los que plantean las relaciones entre clases, las combinan y forman estructuras mayores. Tratan de conseguir que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. A continuación se presenta una breve descripción del patrón estructural a utilizar el desarrollo de la solución.

- **Fachada**

Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. (26)

1.12. Pruebas de SW

Las pruebas de software se definen como la actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas (configuración de la prueba), registrándose los resultados. Se realiza un proceso de evaluación en el que los resultados obtenidos se comparan con los resultados esperados para localizar fallos en el software y se especifican mediante la realización de casos de pruebas como productos de desarrollo de software que ayudan a validar y verificar el sistema como un todo.

Métodos de pruebas

Existen fundamentalmente dos enfoques de prueba que permiten lograr mayor fiabilidad en el software y proporcionan distintos criterios para generar casos de prueba que provoquen fallos en el programa:

- Pruebas de Caja Negra (o Pruebas Funcionales)
- Pruebas de Caja Blanca (o Pruebas Estructurales)

Estas técnicas se pueden aplicar en cualquiera de los niveles de pruebas (unitarias, integración, aceptación, sistema) con diferentes grados de abstracción en la definición de los casos de prueba. (27)

1.12.1. Pruebas de Caja Negra

Las pruebas de Caja Negra se centran principalmente en los requisitos funcionales del software. Se realizan sobre la interfaz, sin considerar la estructura interna del componente que se prueba, simulando una “Caja Negra” cuyo comportamiento sólo puede ser determinado por sus entradas y las salidas obtenidas a partir de ellas. Estas pruebas permiten detectar incorrectas o incompletas funcionalidades en el software, problemas de rendimiento, errores de accesos y de estructuras de datos externas.

Para desarrollar las pruebas de Caja Negra existen varias técnicas:

- Técnica de la Partición de Equivalencia: Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

- Técnica de Grafos de Causa-Efecto: Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. (27)

1.12.2. Pruebas de Caja Blanca

Las pruebas de Caja Blanca se basan en un minucioso examen de los detalles procedimentales del código a evaluar. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Además, permiten comprobar los caminos lógicos del software proponiendo casos de pruebas que ejerciten conjuntos específicos de condiciones y/o bucles. El objetivo de esta técnica es diseñar casos de pruebas para que se ejecuten, al menos una vez, todas las sentencias del programa.

Entre los métodos de prueba de Caja Blanca se encuentran:

- Prueba de Condición: Ejercita las condiciones lógicas contenidas en el módulo de un programa.
- Prueba de Flujo de Datos: Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- Prueba de Bucles: Se centra exclusivamente en la validez de las construcciones de bucles.
- Prueba del Camino Básico: Permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. (27)

1.13. Conclusiones

La investigación realizada permitió sentar las bases para el diseño de la propuesta de solución debido que:

- Se realizó un análisis de sistemas financieros contables que contemplan la gestión de crédito como uno de sus procesos, el cual permitió determinar las principales características de los mismos que pueden ser implementadas para el módulo en desarrollo en aras de hacer más factible la solución.
- Se describieron las principales ventajas y características de las herramientas y tecnologías seleccionadas, las cuales permitieron sentar las bases para diseñar, implementar y validar la solución que se propone.

- Se realizó una valoración del Modelo de Desarrollo definido por el centro CEIGE, determinando los principales elementos que guiaron el proceso de desarrollo del módulo Gestión de Créditos.
- Se describió el estilo arquitectónico que compone el módulo, así como las principales características del patrón arquitectónico Modelo Vista Controlador y los patrones de diseño que conformarán dicho estilo alcanzando así un mayor entendimiento de los mismos.
- Se describen, a grandes rasgos, las pruebas de Caja Negra y Caja Blanca mediante las cuales se validará la implementación a realizar.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1. Introducción

Este capítulo describe la solución propuesta, mostrando las características del sistema para así efectuar el diseño e implementación del mismo, se definen cuáles son los requisitos que presentan problemas así como los puntos de integración que debe tener el módulo y se valida el diseño propuesto mediante métricas aplicadas para determinar si este presenta una calidad aceptable.

2.2. Valoración del análisis

Se realizó una revisión detallada a las especificaciones de requisitos así como a los artefactos generados en el proceso de análisis de software y no se detectaron deficiencias, omisiones, ni errores en los mismos. Los requisitos funcionales descritos por los analistas del sistema facilitaron el entendimiento de los procesos a implementar permitiendo una buena comprensión del problema y posibilitando la identificación de las clases y las funcionalidades que intervienen en el proceso.

2.3. Procesos de Solicitud y Aprobación

Los procesos solicitud y aprobación son de vital importancia para la gestión de créditos debido a que abarcan las características principales, sin las cuales sería imposible llevar a cabo esta actividad. Estos procesos determinan los aspectos y condiciones a tener en cuenta, que componen los créditos que se definan.

Solicitud: es el proceso donde la entidad que necesita el préstamo de un crédito (prestatario) presenta una solicitud de crédito a la entidad que prestará el mismo (prestamista), con un propósito específico en dependencia de la necesidad mediante la cual se realizó la petición.

Aprobación: es el proceso mediante el cual el prestamista después de realizar una evaluación de los estados financieros del prestatario aprueba el crédito solicitado por este, y se procede a aceptar por ambas partes los términos y condiciones formalizados en el contrato legal.

A continuación se muestra en las tablas 2 y 3 la relación de los requisitos de los procesos Solicitud y Aprobación respectivamente, que presentan deficiencias, así como los problemas que traen para la aplicación.

Requisito/s implicado/s.	Integración con Créditos.	Integración con Cobros y Pagos.	Integración con Banco.	Integración Externa.	Continuidad en el desarrollo.
RF: Adicionar tipo de crédito.					x
RF: Modificar tipo de crédito.					x
RF: Eliminar tipo de crédito.	x				x
RF: Listar tipo de crédito.	x				x
RF: Listar solicitud de crédito.	x				x
RF: Eliminar solicitud de crédito.					x

RF: Modificar solicitud de crédito.	x				x
RF: Rechazar solicitud de crédito.	x				x

Tabla #2: Deficiencias encontradas en el proceso Solicitud

Requisito implicado.	**Integración con Créditos.**	**Integración con Cobros y Pagos.**	**Integración con Banco.**	**Integración Externa.**	**Continuidad en el desarrollo.**
RF: Adicionar oferta.	x		x		X
RF: Modificar oferta.	x		x		X
RF: Adicionar contrato.	x		x	X	X
RF: Adicionar contrato.					X

RF: Modificar contrato.					X
RF: Listar contratos.	x				X
RF: Listar contratos.	x			x	x
RF: Listar contratos.	x	x		x	X
RF: Listar contratos.	x	x		x	X
RF: Cancelar contrato					X
RF: Adicionar una solicitud de restructuración o renegociación.	x				X

Tabla #3: Deficiencias encontradas en el proceso Aprobación

2.4. Diseño del sistema

El diseño, es una representación significativa de la ingeniería del producto que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente, y al mismo tiempo la calidad se puede evaluar y cotejar con el conjunto de criterios predefinidos para obtener un buen diseño. En el contexto de la ingeniería del software, el diseño se centra en cuatro áreas importantes de interés: datos, arquitectura, interfaces y componentes. (28)

2.4.1. Patrones de Diseño Utilizados

Los principios del diseño necesarios para construir eficazmente un software pueden codificarse, explicarse y aplicarse de modo metódico, utilizando los denominados patrones de diseño. Cada patrón trata de un

problema específico, que se repite en el diseño o implementación de un sistema de software. Un patrón no es más que una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos, en otras palabras, “los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software”. (29)

Para el desarrollo de la solución propuesta se emplean los siguientes patrones de diseño:

- **Experto:** Se puso en práctica en todos los componentes, con el uso de clases que poseen responsabilidades específicas a cumplir de acuerdo con la información que manejan, los componentes cuentan con clases controladoras, modelos y entidades que poseen funciones concretas de acuerdo con la información que gestionan. Además, se modeló una clase entidad por cada tabla de la Base de Datos, posibilitando el trabajo específico y directo con el experto en la información.

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

- **Controlador:** Se utilizan cuando la aplicación es muy extensa, de esta forma, en el módulo en vez de tener un solo controlador y saturarlo, se tienen clases Controllers, que son controladores más pequeños especializados en las funcionalidades de cada componente.

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

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

- **Fachada:** Su aplicación posibilita promover un débil acoplamiento entre los diferentes componentes que conforman el módulo y su integración con otros subsistemas. Se usa una única clase Service por componente que proporciona los servicios necesarios publicados en el patrón Inversión de Control (IoC) donde se implementan todas las funcionalidades que el componente es capaz de brindar.

- **Inyección de Dependencia:** Este es un patrón de diseño, en el que se suministran objetos a una clase, en lugar de ser la propia clase quien cree los objetos. Este se utiliza en todas las clases controladoras y se evidencia cuando se obtienen las peticiones en los controladores, al hacer las operaciones de la Base de Datos con doctrine.

2.4.2. Diagramas del diseño

Los diagramas del diseño tienen como objetivo crear los modelos que permitan un entendimiento visual de los requisitos del sistema y preparar el ambiente para su implementación y prueba, además su lenguaje es mucho más técnico y cercano a la programación, puesto que son justamente los programadores los usuarios finales de los mismos.

Los diagramas necesarios para la fase de diseño de la solución propuesta y establecidos en el modelo de desarrollo del centro son:

1. Diagrama de paquetes.
2. Diagrama de clases con estereotipos web.
3. Diagramas de secuencia.
4. Diagrama de clases persistentes.
5. Diagrama Entidad-Relación.

A continuación se muestran ejemplos de los diagramas realizados.

2.4.3. Diagrama de paquetes

El diagrama de paquetes es utilizado para mostrar las agrupaciones lógicas en las que se encuentra dividido un sistema y las dependencias entre ellas. Suministran una descomposición de la jerarquía lógica

de un sistema. A continuación se muestra el diagrama de paquetes del módulo Gestión de Créditos donde se modelan los cuatro componentes que lo componen.

- **Configuración:** Este componente tiene encapsuladas todas las clases que le dan solución a las configuraciones previas para llevar a cabo la Gestión de los Créditos.
- **Administración:** Este componente tiene encapsuladas todas las clases que le dan solución a los procesos involucrados en la Gestión de Créditos.
- **Carga inicial:** Este componente tiene encapsuladas todas las clases que permiten realizar la carga inicial de las informaciones necesarias para poder realizar el proceso de Gestión de Créditos.
- **Recuperaciones:** Este componente tiene encapsuladas todas las clases necesarias para realizar los reportes de las informaciones en el sistema.

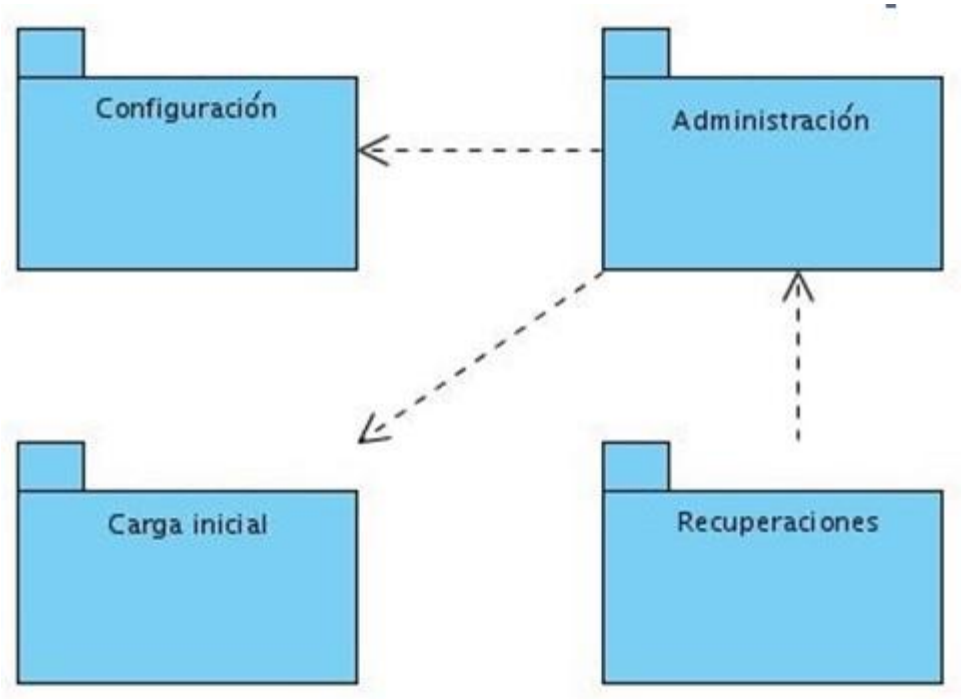


Figura #1: Diagrama de paquetes

2.4.4. Diagramas de clases del diseño con estereotipos web

Los Diagramas de Clases del Diseño con Estereotipos Web representan una idea general de la navegación entre páginas de la aplicación web. Este se debe crear por subsistema. Sirve de gran ayuda para identificar con mucha rapidez la ubicación de las opciones dentro del sistema y de las características de cada subsistema. En la figura #2 se presenta el diagrama correspondiente al requisito funcional *Gestionar Solicitud de Crédito* donde se observa la clase *gestsolicitud.phtml* que contiene componentes generados en la librería JavaScript ExtJs y es responsable de visualizar a través de *gestsolicitud.js,* la información necesaria para gestionar las solicitudes de créditos. Esta clase a la vez incluye la controladora con igual nombre que contiene la implementación de las funcionalidades, y hereda de *ZendExt_Controller_Secure* para gestionar la seguridad. Se representan también las clases que forman parte de la capa lógica del negocio y de las cuales se nutren las clases controladoras.

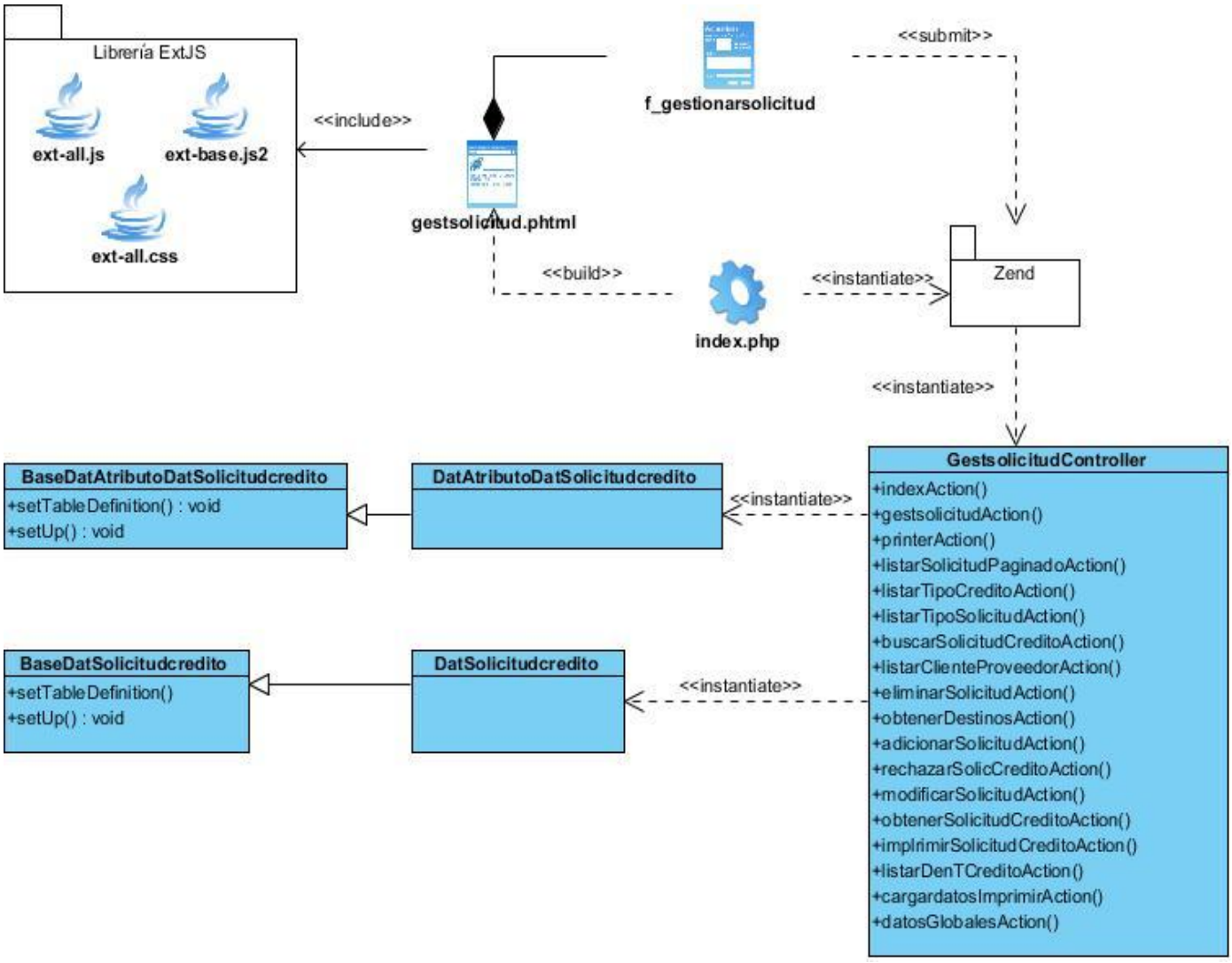


Figura #2: Diagrama de clases del diseño con estereotipos web del requisito Gestionar Tipo de Crédito

Los restantes diagramas de clases del diseño con estereotipos web se encuentran en el ANEXO 1.

2.4.5. Diagrama de secuencia

Este diagrama se encarga de modelar la interacción entre objetos durante el ciclo de ejecución de un proceso. Es decir, la secuencia de llamadas entre objetos en un periodo de tiempo. La figura #3 muestra el diagrama de secuencia del requisito funcional *Adicionar Solicitud de Crédito*, perteneciente al componente de *Administración*, donde se puede observar que el mismo se inicia cuando el usuario

introduce los datos de la solicitud de crédito, luego estos datos son enviados a la clase controladora que valida si los mismos son correctos y los inserta en la Base de Datos correspondiente, luego el sistema muestra un mensaje confirmando que la solicitud de crédito fue insertada satisfactoriamente.

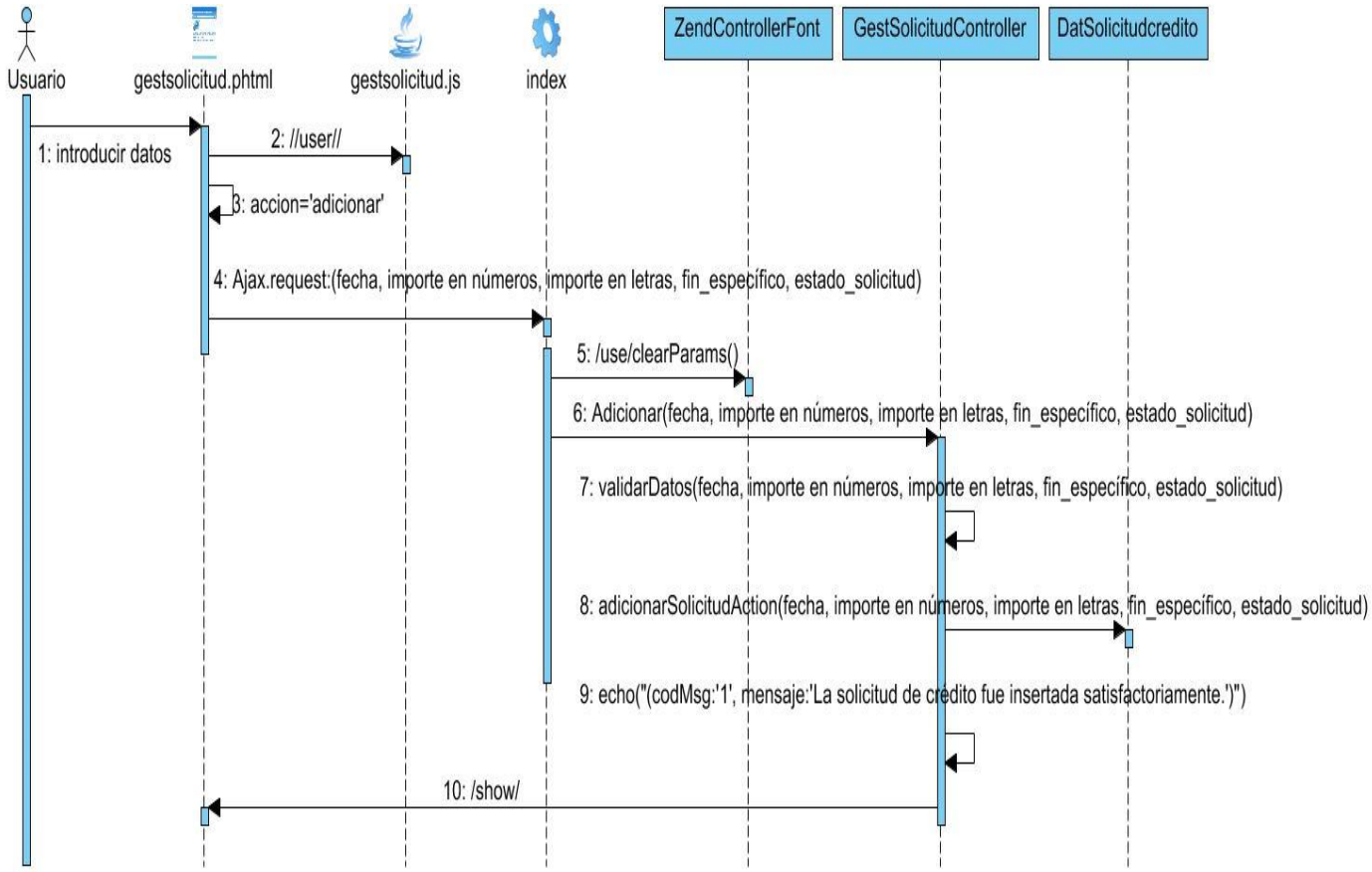


Figura #3: Diagrama de Secuencia del requisito Adicionar Tipo de Crédito

Los restantes diagramas de secuencia se encuentran en el ANEXO 2.

2.4.6. Diagrama de clases persistentes

El diagrama de clases persistentes, describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información: clases, asociaciones y atributos, interfaces, con sus operaciones y constantes, métodos, información sobre los tipos de atributos, navegabilidad y dependencias. En la figura #4 se muestra el diagrama de clases persistentes del requisito

Gestionar Solicitud de Crédito, perteneciente al componente *Administración*, representando un conjunto de clases, atributos y las relaciones entre ellas.

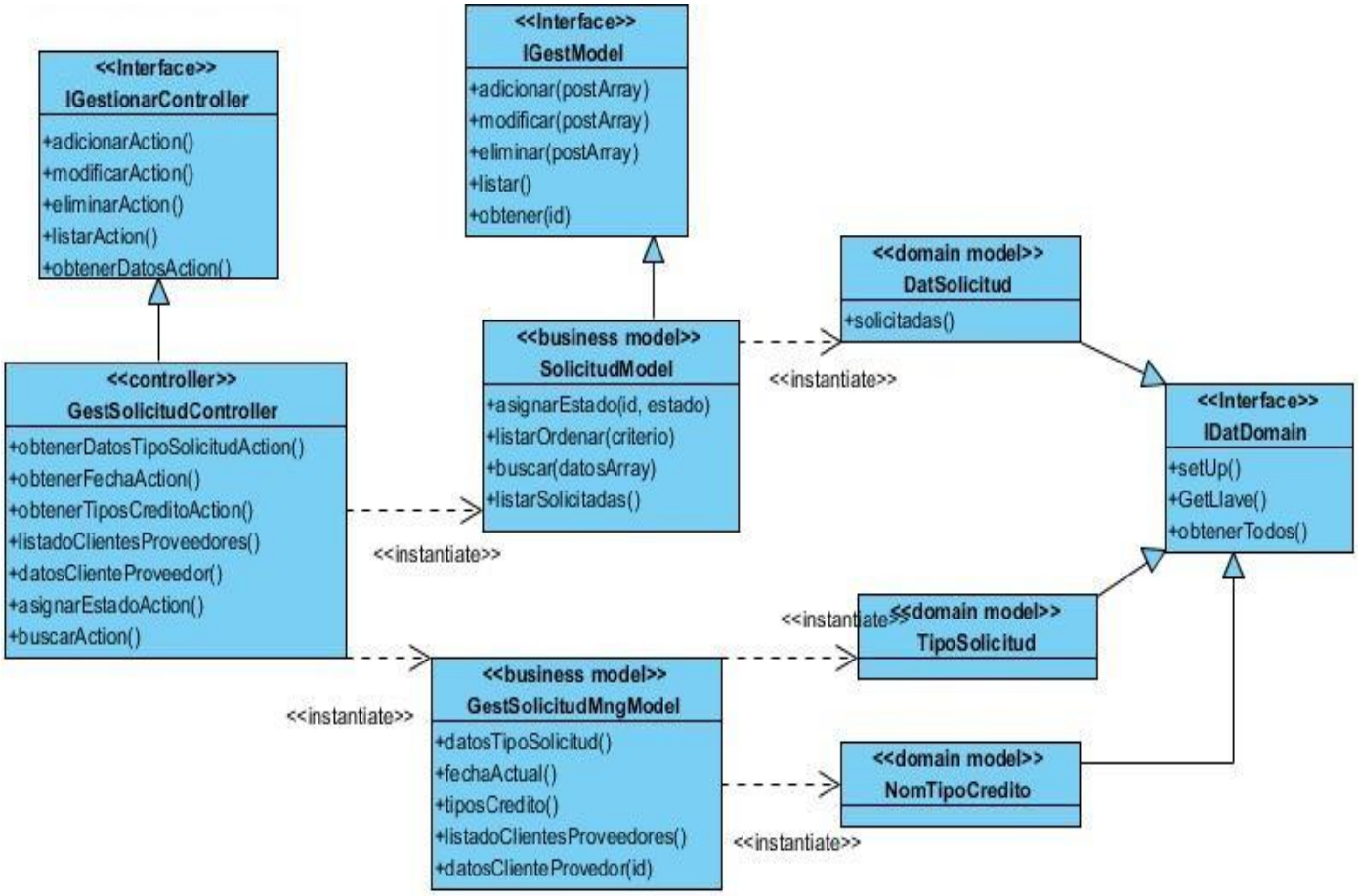


Figura #4: Diagrama de Clases Persistentes del Requisito Gestionar Solicitud

El resto de los diagramas de clases persistentes del diseño se encuentran en el ANEXO 3.

2.4.7. Diagrama Entidad-Relación (DER)

El DER, se centra en buscar las entidades básicas del modelo y luego la relación que existe entre ellas. En la figura #5, se muestra un fragmento del DER, correspondiente al módulo Gestión de Créditos, que consta de 11 entidades, creado para lograr un correcto diseño de la Base de Datos. El diagrama original consta de 31 entidades y se presenta en el ANEXO 4.

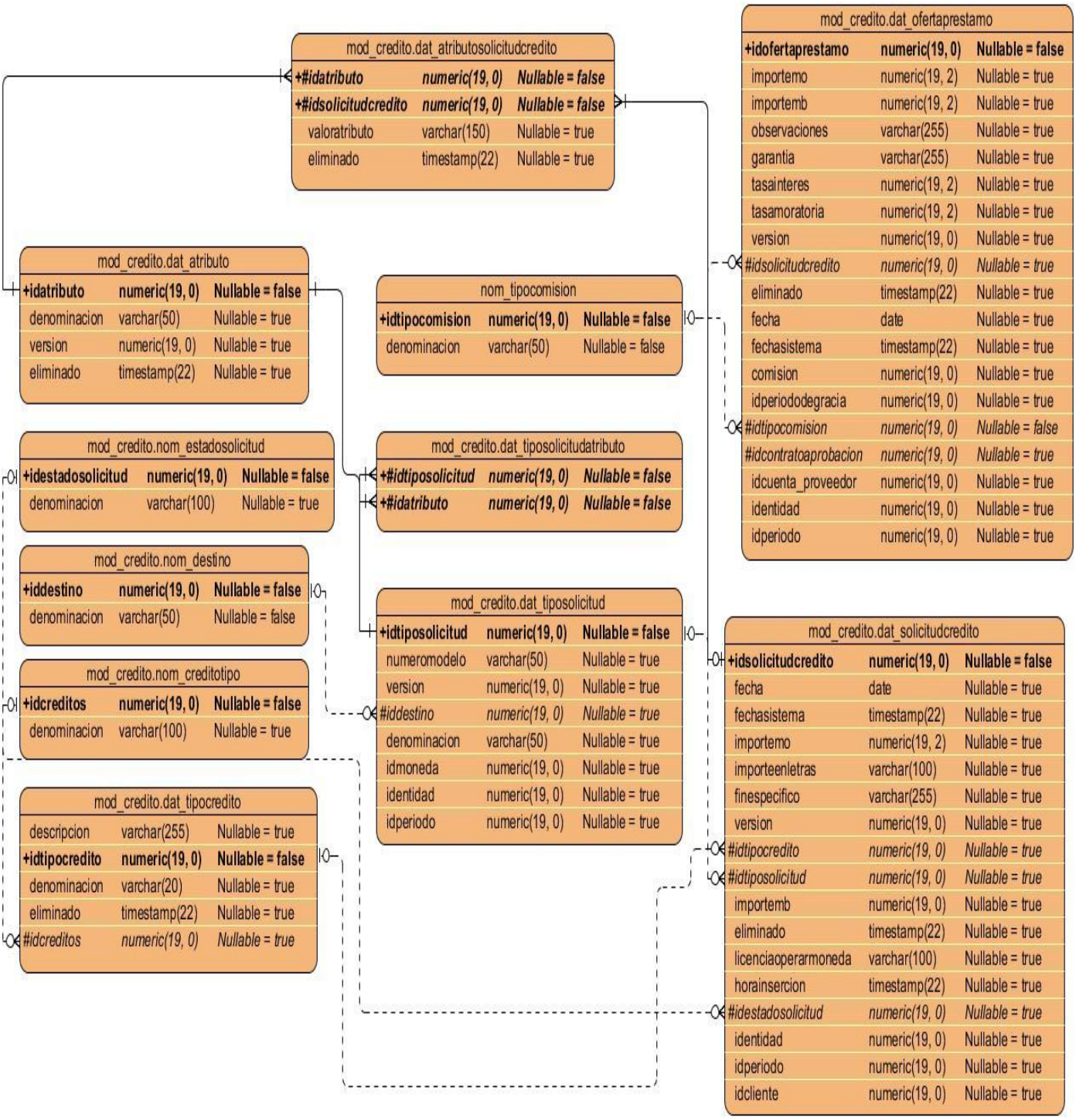


Figura #5: Fragmento del Diagrama Entidad-Relación del módulo Gestión de Créditos

La siguiente tabla describe las entidades con las cuales se trabaja para darle solución a los problemas presentados en el módulo Gestión de Créditos:

Nombre	Descripción
dat_restructuracionrenegociacion	Tabla que almacena las reestructuraciones y renegociaciones.
nom_restructuracionrenegociacion	Nomenclador que almacena los tipos de restructuraciones o renegociaciones.
nom_tipocomision	Nomenclador que almacena los tipos de comisiones.
nom_conceptoderecho	Tabla que almacena los tipos de derechos
nom_periododegracia	Tabla que almacena todos los periodos
nom_duracioncontrato	Tabla que almacena la duración de los contratos
dat_cancelacion	Tabla que almacena todas las cancelaciones.
dat_contratoderecho	Tabla que almacena idcontrato,idderecho y el tipo de derecho
nom_estadocontratoaprobacion	Nomenclador que almacena los nombres de los estados de aprobación.
dat_intereses	Tabla que almacena los intereses.
dat_contratoaprobacion	Tabla que almacena todos los contratos aprobados.
dat_tipocredito	Nomenclador que almacena el tipo de crédito.
nom_creditotipo	Nomenclador que almacena los tipos de créditos.
dat_ofertaprestamo	Tabla que almacena las ofertas de préstamo.
nom_estadosolicitud	Tabla que almacena los estados de las solicitudes.
nom_destino	Nomenclador que almacena los destinos.
dat_atributosolicitudcredito	Tabla que almacena los identificadores de atributos y solicitudes de créditos para un valor.
dat_tiposolicitud	Tabla que almacena los tipos de solicitudes.
dat_operacionnotificacion	Tabla que almacena todas las operaciones que se generan por un derecho u obligación de pago.

dat_operacionreal	Tabla que almacena las operaciones reales
dat_atributo	Tabla que almacena los atributos definidos.
dat_solicitudcredito	Tabla que almacena las solicitudes de créditos.
dat_amortizacion	Tabla que almacena las amortizaciones.
dat_tiposolicitudatributo	Tabla que almacena los identificadores de atributos y solicitudes de créditos además del valor de su atributo.
nom_tipooperacion	Nomenclador que almacena los tipos de operaciones.
dat_operacionesplazos	Tabla que almacena las operaciones conjuntamente con los plazos y sus contratos de aprobación
dat_planificacion	Tabla que almacena las planificaciones
nom_estadooperacionreal	Nomenclador que almacena los estados de operaciones
nom_plazos	Nomenclador que almacena los plazos.

Tabla #4: Entidades de la Base de Datos

2.5. Métricas para la evaluación del diseño

Una métrica, es cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software o un sistema de información, generalmente para realizar comparativas o para la planificación de proyectos de desarrollo. Para medir la calidad del diseño del presente trabajo, se empleó las métricas: Tamaño Operacional de Clases (TOC) y Relaciones entre clases (RC). (30)

2.5.1. La métrica TOC

Esta métrica es propuesta por Lorenz y Kidd, en su libro sobre métricas Orientada a Objetos (OO) y se centra en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema como un todo. Evalúa los siguientes atributos de calidad:

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

Tabla #5: Evaluación de los atributos de calidad, métrica TOC

Para aplicar la métrica, son necesarios los valores de los umbrales para los parámetros de calidad. Los umbrales aplicados fueron:

	Categoría	Criterio
Complejidad de Mantenimiento	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Tabla #6: Valores de los umbrales para la métrica TOC

Para aplicar la métrica TOC se define un listado con los procedimientos que intervienen en el proceso:

No	Componente	Clase	Cantidad de Procedimientos
1	Gestión de Créditos	ConfigtiposolicitudController	11
2	Gestión de Créditos	TiposolicitudModel	12
3	Gestión de Créditos	GesttipocreditoController	7
4	Gestión de Créditos	TipocreditoModel	9
5	Gestión de Créditos	GestsolicitudController	14
6	Gestión de Créditos	SolicitudModel	10
7	Gestión de Créditos	GestofertaController	17
8	Gestión de Créditos	OfertaModel	13
9	Gestión de Créditos	GestcontratoController	46
10	Gestión de Créditos	ContratoaprobacionModel	47

11	Gestión de Créditos	GestreestructuracionrenegociacionController	15
12	Gestión de Créditos	DatRestructuracionrenegociacionModel	17

Tabla #7: Listado de las clases con su cantidad de procedimientos

Se determina el promedio de los procedimientos que intervienen en el proceso con respecto a la cantidad de clases.

Total de clases: 12

Promedio de procedimientos (Sumatoria de los procedimientos entre la cantidad de clases): 18.16666667

Teniendo en cuenta el promedio obtenido, el rango de responsabilidad, la complejidad y la reutilización de cada clase queda como se muestra en la tabla #6:

Responsabilidad	Complejidad	Reutilización
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Baja	Baja	Alta
Alta	Alta	Baja
Alta	Alta	Baja
Baja	Baja	Alta
Baja	Baja	Alta

Tabla #8: Rango de los atributos que intervienen en el proceso de la métrica TOC

La métrica TOC arrojó los siguientes resultados:

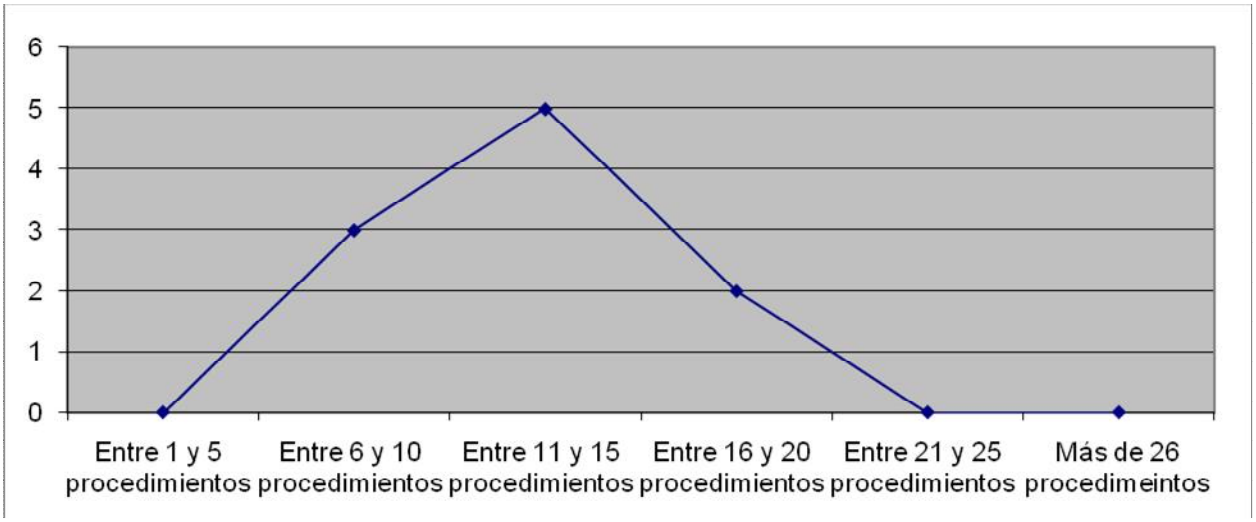


Figura #7: Resultados de la evaluación de la métrica TOC agrupados por la tendencia de los valores

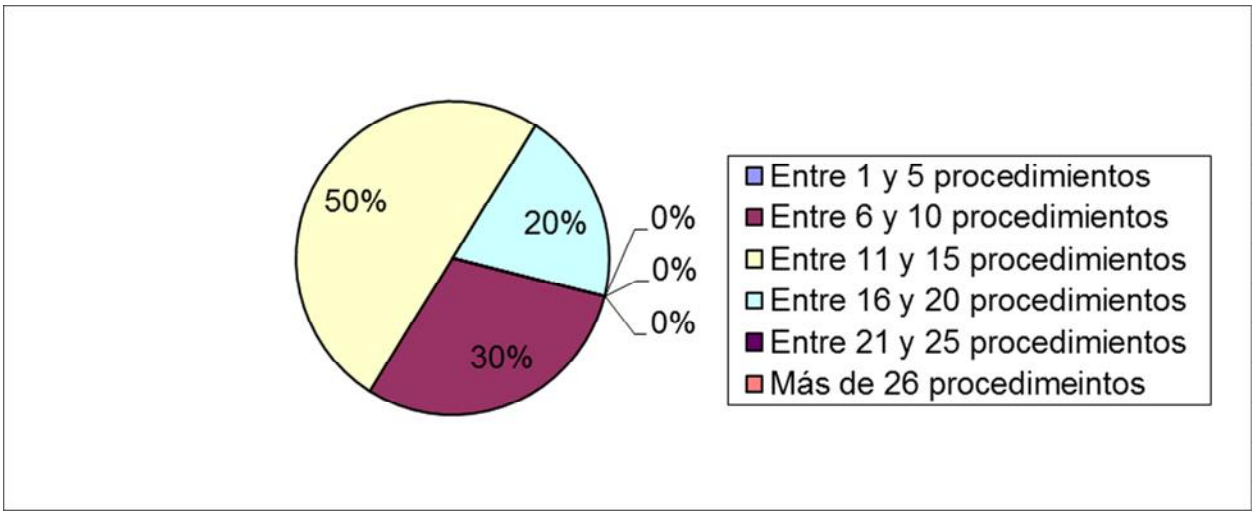


Figura #6: Resultados por criterios de dependencia

- **Responsabilidad:**

Responsabilidad	Cantidad de clases	Promedio
Baja	10	83.33333333
Media	0	0
Alta	2	16.66666667

Tabla #9: Resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

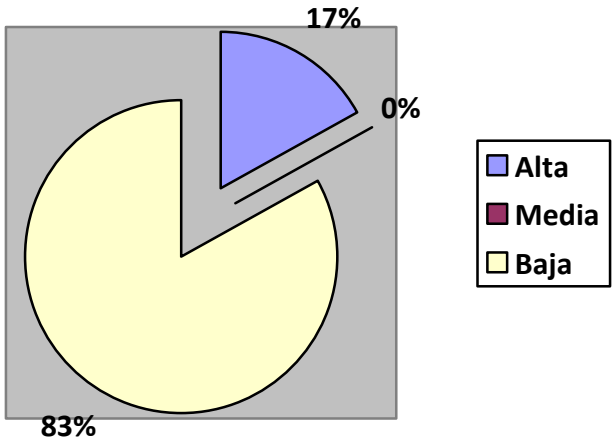
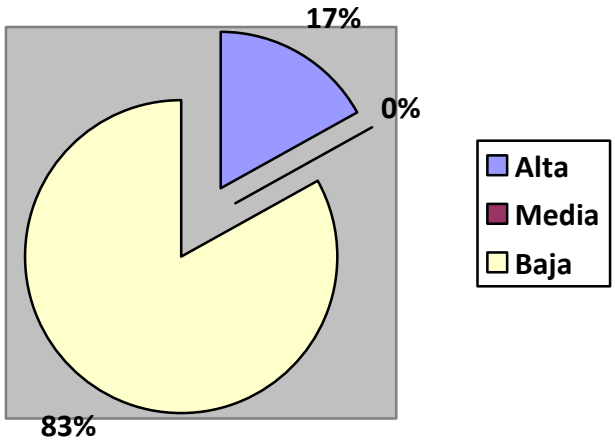


Figura #7: Resultados gráficos de la evaluación de la métrica TOC en el atributo Responsabilidad

- **Complejidad:**

Complejidad	Cantidad de clases	Promedio
Baja	10	83.33333333
Media	0	0
Alta	2	16.66666667

Tabla #10: Resultados de la evaluación de la métrica TOC en el atributo Complejidad

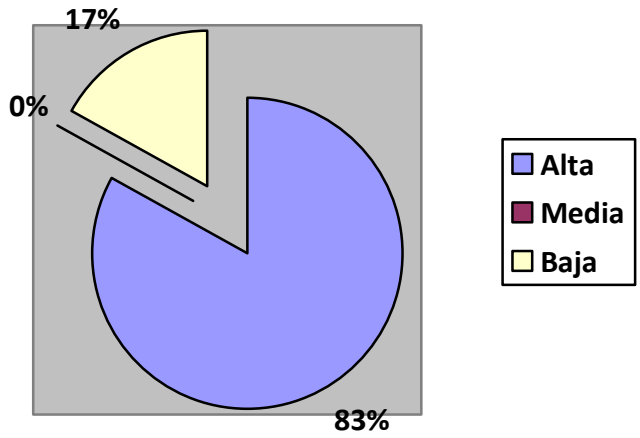


Figura#8: Resultados gráficos de la evaluación de la métrica TOC en el atributo Complejidad

- **Reutilización:**

Reutilización	Cantidad de clases	Promedio
Alta	10	83.33333333
Media	0	0
Baja	2	16.66666667

Tabla #11: Resultados de la evaluación de la métrica TOC en el atributo Reutilización



Figura#9: Resultados gráficos de la evaluación de la métrica TOC en el atributo Reutilización

Con la utilización de esta métrica para evaluar el diseño se obtuvo como resultado que el tamaño promedio de las operaciones por clases es bajo. Esto implica que la complejidad de la implementación y la reutilización de las mismas sean bajos y las clases existentes se puedan reutilizar ampliamente.

2.5.2. La métrica RC.

Las relaciones entre las clases están dadas por el número de relaciones de uso que tenga una clase con otras. Mediante la cual se calcula el Acoplamiento, la Complejidad, la Reutilización y la Cantidad de pruebas a fin de inspeccionar la efectividad del diseño.

Atributos que afecta:	Modo en que lo afecta:
Acoplamiento	El aumento de la RC provoca un aumento del acoplamiento de la clase.
Complejidad del mantenimiento	El aumento de la RC provoca un aumento de la complejidad del mantenimiento de la clase.
Reutilización	El aumento de la RC provoca una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	El aumento de la RC provoca un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Tabla #12: Atributos de calidad

Para aplicar la métrica, son necesarios los valores de los umbrales para los parámetros de calidad. Los umbrales aplicados fueron:

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad Mantenimiento	Baja	<= Prom.

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

Tabla #13: Valores de los umbrales para la métrica RC

Para el cálculo de la métrica RC primeramente se listan los procedimientos de las clases que intervienen en el proceso.

No	Módulo	Clase	Cantidad de Relaciones de Uso
1	Gestión de Créditos	ConfigtiposolicitudController	1
2	Gestión de Créditos	TiposolicitudModel	1
3	Gestión de Créditos	GesttipocreditoController	1
4	Gestión de Créditos	TipocreditoModel	1
5	Gestión de Créditos	GestsolicitudController	2
6	Gestión de Créditos	SolicitudModel	2
7	Gestión de Créditos	GestofertaController	2
8	Gestión de Créditos	OfertaModel	2
9	Gestión de Créditos	GestcontratoController	4
10	Gestión de Créditos	ContratoaprobacionModel	3
11	Gestión de Créditos	GestreestructuracionrenegociacionController	2
12	Gestión de Créditos	DatRestructuracionrenegociacionModel	2

Tabla #14: Listado de las relaciones de uso de las clases que intervienen en el proceso de la métrica RC

Se le halla el promedio de sus asociaciones de uso con respecto a la cantidad de clases:

Total de clases: 12

Promedio de asociaciones de uso (Sumatoria de la cantidad de Relaciones de Uso entre la cantidad de clases): 1.916666667

Se obtiene el rango de Acoplamiento, Complejidad, Reutilización y Cantidad de pruebas de las relaciones:

Acoplamiento	Complejidad Mantenimiento.	Reutilización	Cantidad de Pruebas
Baja	Baja	Baja	Alta
Baja	Baja	Baja	Alta
Baja	Baja	Baja	Alta
Baja	Baja	Baja	Alta
Media	Media	Media	Media
Media	Media	Media	Media
Media	Media	Media	Media
Media	Media	Media	Media
Alta	Alta	Alta	Baja
Alta	Media	Media	Media
Media	Media	Media	Media
Media	Media	Media	Media

Tabla #15: Rango de los atributos que intervienen en el proceso de la métrica RC

A continuación se muestran los resultados de aplicar la métrica RC.

Criterio	Categoría	Cantidad de clases	Promedio
0 dependencias	Muy Bueno	0	0
1 dependencias	Bueno	4	33.33333333
2 dependencias	Regular	6	50
3 dependencias	Malo	1	8.333333333
> 3 dependencias	Muy Malo	1	8.333333333

Total		12	100

Tabla #16: Resultados por criterios de dependencia

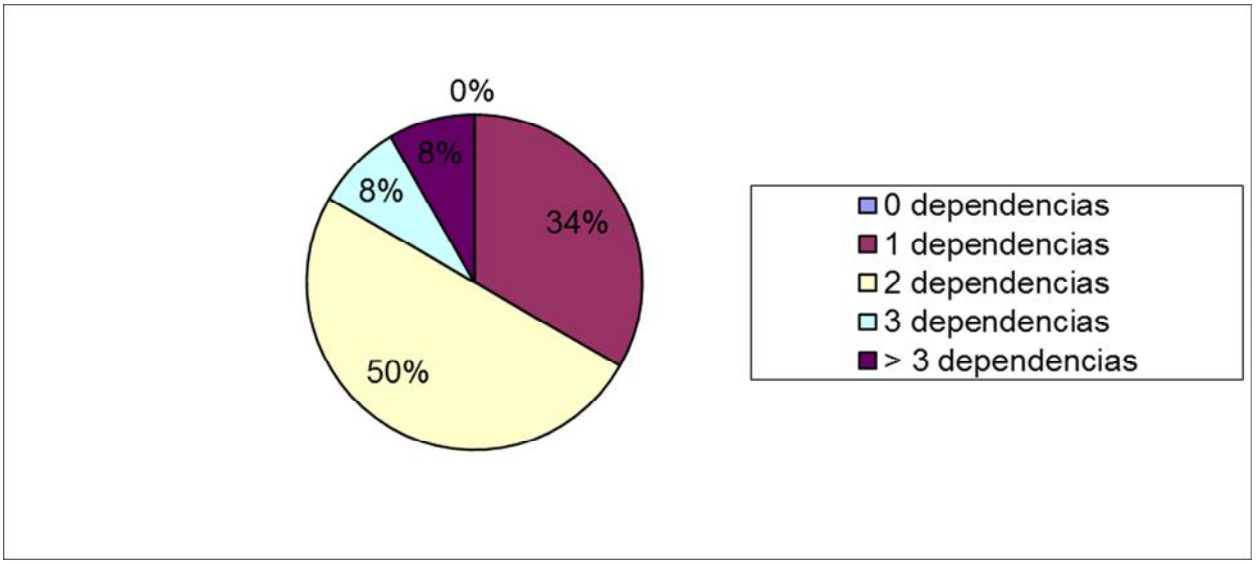


Figura #10 Resultados por criterios de dependencia.

- **Acoplamiento:**

Acoplamiento	Cantidad de clases	Promedio
Ninguno	0	0
Bajo	4	33.33333333
Medio	6	50
Alto	2	16.66666667

Tabla #17: Resultados de la evaluación de la métrica RC en el atributo Acoplamiento

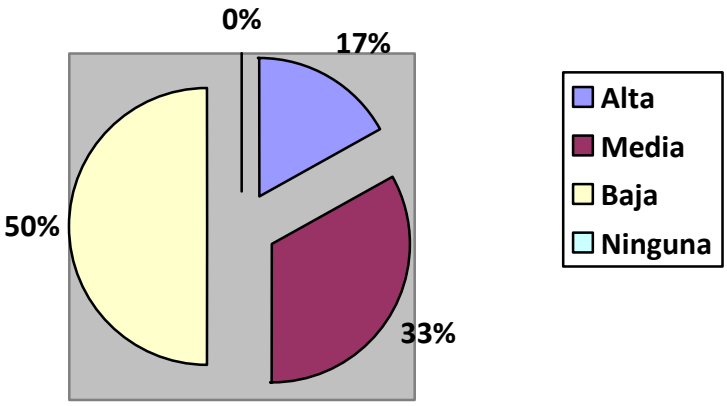


Figura #11: Resultados gráficos de la evaluación de la métrica RC en el atributo Acoplamiento

- **Complejidad de Mantenimiento:**

Complejidad de Mantenimiento	Cantidad de clases	Promedio
Baja	4	33.33333333
Media	7	14.28571429
Alta	1	7.142857143

Tabla #19: Resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

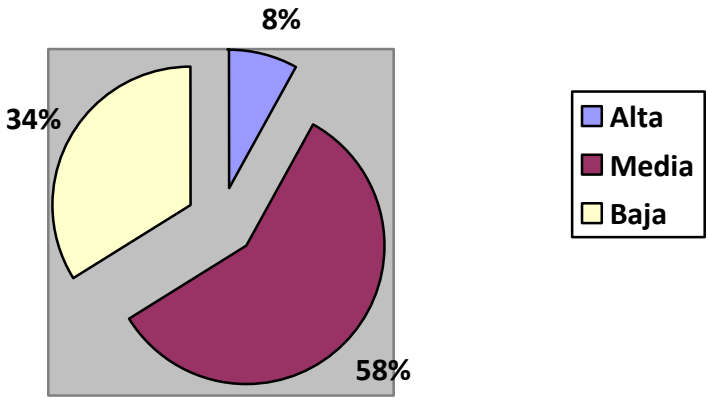


Figura #12: Resultados gráficos de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

- **Cantidad de Pruebas:**

Cantidad de Pruebas	Cantidad de clases	Promedio
Baja	4	33.33333333
Media	7	58.33333333
Alta	1	8.333333333

Tabla #20: Resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

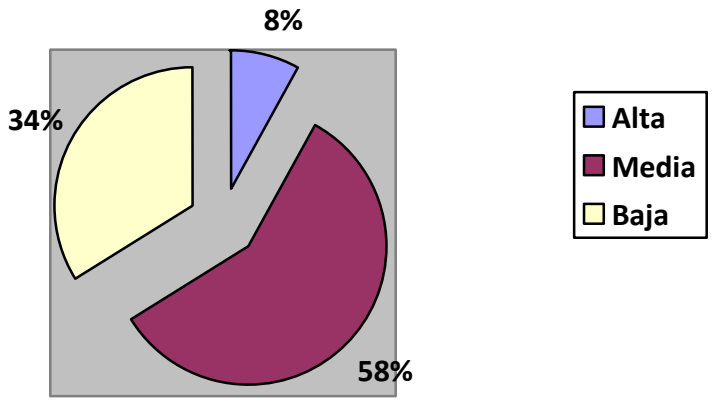


Figura #13: Resultados gráficos de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

- **Reutilización:**

Reutilización	Cantidad de clases	Promedio
Baja	1	8.333333333
Media	7	58.33333333
Alta	4	33.33333333

Tabla #21: Resultados de la evaluación de la métrica RC en el atributo Reutilización.

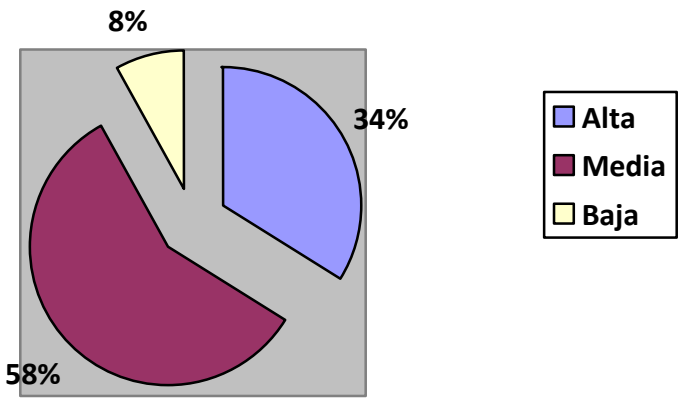


Figura #14: Resultados gráficos de la evaluación de la métrica RC en el atributo Reutilización.

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que el diseño propuesto para el componente Gestión de Créditos se encuentra dentro de los niveles de calidad requeridos. Los atributos de calidad fueron evaluados satisfactoriamente confirmando la elevada reutilización, el bajo acoplamiento, la baja complejidad y la baja cantidad de pruebas que se necesitan realizar en el diseño propuesto.

2.6. Conclusiones

Al concluir este capítulo, quedó definido el diseño de la solución propuesta teniendo en cuenta que:

- Se identificaron los requisitos a modificar para darle respuesta a la propuesta de solución descrita, ayudando así a comprender las acciones a desarrollar, lo que permitió lograr un mejor entendimiento del negocio en cuestión.
- Se describieron los patrones de diseño a utilizar, determinándose así la manera en que se relacionarán las clases implicadas en el desarrollo de la solución propuesta.
- Se realizaron los diagramas de clases de diseño, secuencia, paquetes y el diagrama de entidad relación para lograr un mejor entendimiento de la implementación de los requisitos.
- Se validó el diseño propuesto para el desarrollo del módulo Gestión de Créditos mediante la aplicación de las métricas TOC y RC, determinándose que el mismo se encuentra en los niveles aceptables de calidad.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3.1. Introducción

En este capítulo se realiza la validación a la solución propuesta, con el objetivo de comprobar el correcto funcionamiento de las clases u operaciones utilizadas, además se describen los estándares de codificación a utilizar, los cuales son definidos por la dirección del centro CEIGE así como la manera en que se desplegará la solución.

3.2. Estándar de codificación

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. Debido a la complejidad del sistema CEDRUX, el numeroso personal involucrado en él y el alto nivel de integración existente entre sus componentes, el grupo arquitectónico del proyecto definió normas de codificación con el fin de obtener un estándar en la implementación por el equipo de desarrollo que permitiera asegurar la calidad del software, obteniendo un código más legible y reutilizable. Los estándares de codificación establecerán las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que pueda aumentar su mantenibilidad a lo largo del tiempo. (31)

3.2.1. PascalCasing

El estándar PascalCasing establece que los identificadores, nombres de clases, variables, métodos o funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. La nomenclatura de las clases se realizó sobre la base de este estándar, usando palabras compuestas sugerentes acordes al propósito de la misma. (31)

3.2.2. Nomenclatura de clases según su tipo

- **Controllers**: Clases controladoras del negocio.

El nombre de la clase controladora debe estar estructurado por el nombre propio de la misma en mayúsculas seguido por la palabra Controller y heredar siempre de la super clase del framework ZendExt_Controller_Secure.

- **Clases de los modelos**
  - ✓ **Business**: Clases modelo del negocio.

Las clases modelo tendrán por identificador el nombre de la tabla en la que trabajan seguido por la palabra Model y heredarán de la super clase del framework Zend_Ext llamada ZendExt_Model.

- ✓ **Domain**: Clases entidades del dominio.

Los archivos situados en el domain tienen el mismo nombre de las tablas que representan, definiéndolas como clases php, pueden incluir los prefijos Dat o Nom para diferenciar entre sus usos.

- ✓ **Generated**: Clases bases del dominio

Las clases que se encuentran dentro de Generated comienzan su nombre con la palabra: “Base”, seguido del nombre de la tabla en la Base de Datos.

- ➢ **Validators**: Clases validadoras.

El nombre de la clase validadoras se especifica con nombres compuestos con el distintivo que terminan en Validador.

- ➢ **Services**: Clases que ofrecen los servicios de los componentes.

Estas clases, de acuerdo con las operaciones que realizan y prestaciones que brindan, se definen con calificativos sugerentes, agregándoles la terminación Service. Ejemplo: AdministracionService. (31)

3.2.3. CamelCasing

El estándar CamelCasing es parecido al PascalCasing con la particularidad de que la letra inicial del identificador no comienza con mayúscula. Esta notación se utilizó para el nombre de funciones y atributos.

Nomenclatura de las funciones: El identificativo a emplear para las funciones o métodos se escribe con la primera palabra en minúscula utilizando la notación CamelCasing y nombres que deduzcan su propósito. Ejemplo: cargarCuentaBancaria. Los denominadores de las acciones de las clases controladoras tienen la peculiaridad de ir seguidos por la palabra “Action”. Ejemplo: adicionarCreditoAction.

Nomenclatura de los atributos: Las variables se nombran convenientemente de acuerdo con el estándar CamelCasing. Ejemplo: nombreCliente. (31)

3.3. Diagrama de despliegue

El diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema, en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo.

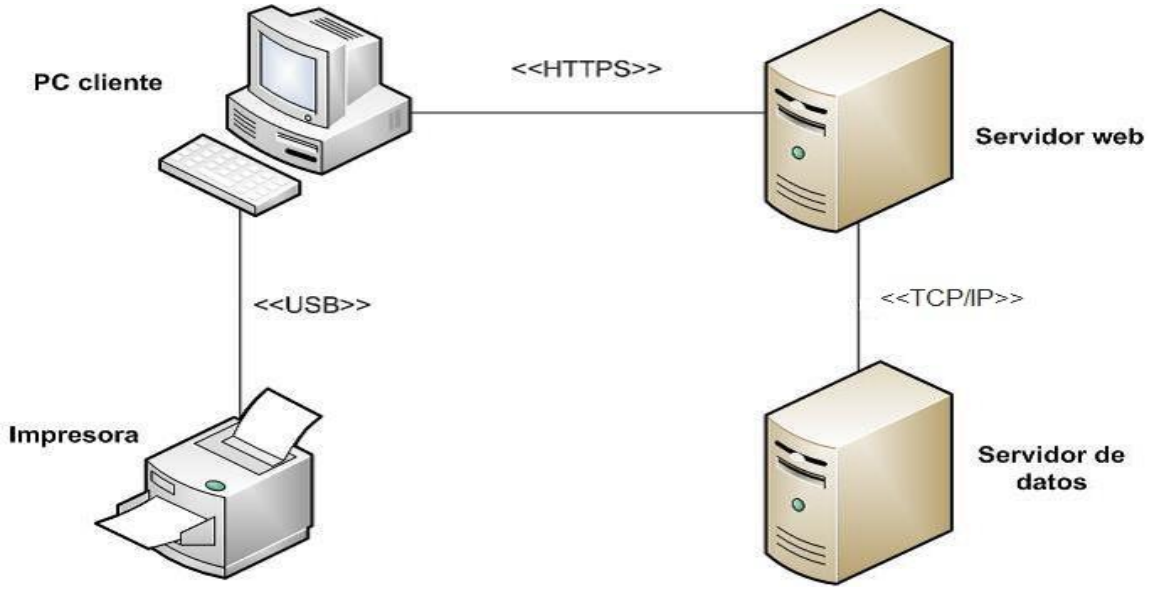


Figura #15: Diagrama de Despliegue

- **Nodo Servidor de Base de Datos**

  En este dispositivo se encuentra instalado el servidor de Base de Datos Central para el control y persistencia física de los recursos y servicios que brinda el sistema. Esto permite guardar un histórico sobre las acciones realizadas en la aplicación en un lapso de tiempo configurable, como las configuraciones del propio sistema que definen reglas en la lógica del negocio en el sistema.

- **Nodo Servidor**

  En este dispositivo se encuentra instalado el Servidor Central de Aplicaciones para la publicación de información en la red (Apache 2). Esto permite crear configuraciones a la medida del sistema mediante aplicaciones y módulos adicionales, que ayudan a mejorar la gestión y el acceso. La posibilidad de acceder de forma segura a la información, definiendo distintos niveles de control, es otra de las ventajas fundamentales de este tipo de servidor de aplicaciones.

- **Nodo PC Cliente**

  EL cliente accede a la aplicación web publicada en el Servidor Central de Aplicaciones mediante el protocolo HTTP. Este nodo provee el acceso de todas las funcionalidades y prestaciones que brinda el sistema como producto según el nivel de acceso y tipo de usuario.

3.4. Pruebas del software

Las pruebas del software, conocidas también como técnicas de evaluación dinámica, son un elemento crítico para la garantía de la calidad del sistema, representan una revisión final de las especificaciones del diseño y de la implementación. Su principal objetivo es diseñar pruebas que sistemáticamente, saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo. (32)

3.4.1. Pruebas funcionales

Las pruebas funcionales están basadas en técnicas de Caja Negra y se llevan a cabo sobre la interfaz del software, con el objetivo de demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto y que la integridad de la información externa se mantiene. La prueba de Caja Negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.(32)

Aplicación de la prueba

Para verificar que la aplicación se comporta según los requerimientos establecidos por el cliente, se diseñan casos de pruebas usando el método de Caja Negra. A continuación se especifica el caso de prueba para el requisito “Eliminar tipo de crédito”.

Condiciones de ejecución:

- Se debe haber adicionado al menos un tipo de crédito.

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Eliminar tipo de crédito	El sistema debe permitir eliminar un tipo de crédito.	EP 1.1: Eliminar tipo de crédito.	1. Se selecciona el tipo de crédito a eliminar. • Se presiona la opción **Eliminar**. • Se presiona el botón **Aceptar**. • Se muestra un mensaje de información. • Se presiona el botón **Aceptar**.
		EP 1.2: Eliminar tipo de crédito que esté siendo usado.	2. Se selecciona el tipo de crédito a eliminar. • Se presiona la opción **Eliminar**. • Se presiona el botón **Aceptar**. • Se verifica si el tipo de crédito está siendo usado por una solicitud. • Se muestra un mensaje de información. • Se presiona el botón **Aceptar**.
		EP 1.3: Cancelar.	3. Se selecciona del tipo de crédito a eliminar. • Se presiona la opción **Eliminar**. • Se presiona el botón **Cancelar**.

Tabla #22: Caso de prueba Eliminar tipo de crédito

Descripción de variable

No	Nombre de campo	Tipo	Válido	Inválido
1	Denominación	Campo de texto.	Cadena de caracteres.	Caracteres especiales y números.

2	Descripción	Campo de texto.	Cadena de caracteres.	Caracteres especiales y números.
3	Tipo de crédito	Campo de selección (No editable).	Cadena de caracteres.	Caracteres especiales y números.

Tabla #23: Descripción de las variables

Juegos de datos a probar:

Id del escenario	Escenario	Denominación	Descripción	Tipo de crédito	Respuesta del sistema
EP 1.1	Eliminar tipo de crédito	N/A	N/A	N/A	El sistema elimina el tipo de crédito y muestra el mensaje de información: “El tipo de crédito fue eliminado correctamente.”. El sistema cierra la interfaz.
EP 1.2	Eliminar tipo de crédito que esté siendo usado.				El sistema muestra el mensaje de información: “El tipo de crédito está siendo usado por una solicitud.”. El sistema cierra la interfaz.
EP 1.3	Cancelar.	NA	NA	NA	El sistema cierra la interfaz sin realizar ninguna operación.

Tabla #24: Juego de datos

Resultados de las pruebas

Las Pruebas Funcionales, como la ejemplificada anteriormente, fueron aplicadas al resto de los requisitos del sistema, realizándose un caso de prueba por cada requisito funcional, los cuales se encuentran especificados en los documentos de: Diseño de Casos de Pruebas, correspondientes a cada uno de los requisitos funcionales. Como resultado de aplicarse estas pruebas, se obtuvo en la primera iteración un total de 5 no conformidades las cuales corresponden a validaciones en la interfaz y validaciones en la lógica del negocio. Para una segunda iteración estas no conformidades fueron resueltas en su totalidad, contribuyendo así al buen funcionamiento del sistema.

3.4.2. Pruebas de Caja Blanca

Las pruebas de Caja Blanca se basan en un minucioso examen de los detalles procedimentales del código a evaluar. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Además, permiten comprobar los caminos lógicos del software proponiendo casos de pruebas que ejerciten conjuntos específicos de condiciones y/o bucles. El objetivo de esta técnica es diseñar casos de pruebas para que se ejecuten, al menos una vez, todas las sentencias del programa. (32)

Aplicación de la prueba

Para el desarrollo de la prueba de Caja Blanca se aplicó la técnica del camino básico al procedimiento “eliminarAction” de la clase GesttipocreditoController.

```
function eliminarAction() {
    $idcredito= $this->_request->getPost("idcredito"); 1
    $tipoCredito = new TipocreditoModel();  1
    $arrTipo = $tipoCredito->obtenerTipo($idcredito); 1
        if( $arrTipo[0]['eliminado'] == null ) { 2
            $eliminado = $tipoCredito->eliminar($idcredito); 3
            if($eliminado) { 4
                echo "{success:true}";  5
            }
            else
                echo "{success:false}"; 6
        }
        else {
            echo "{failure:true}" ; 7
        }
    } 8
```

Figura #16: Flujo del algoritmo eliminarAction

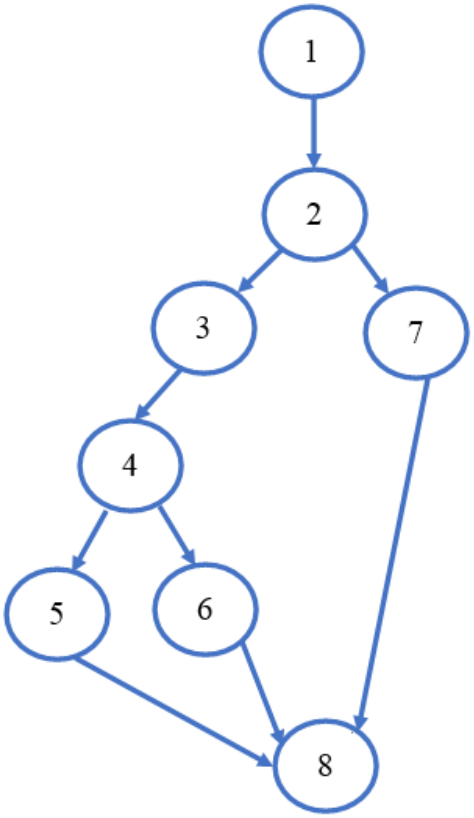


Figura #17: Grafo de flujo asociado al algoritmo

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática, mediante las tres fórmulas siguientes:

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 anterior se obtuvieron los siguientes resultados:

Calculando mediante la fórmula 1:

V (G) = (9 - 8) + 2

V (G) = 3.

Calculando mediante la fórmula 2:

V (G) = 2+ 1

V (G) = 3.

Calculando mediante la fórmula 3:

V (G) = 3.

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 3, lo que significa que existen tres 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:

Camino básico #1	*1 – 2 – 3 – 4 – 5 – 8*
Camino básico #2	*1 – 2 – 3 – 4 – 6 – 8*
Camino básico #3	*1 – 2 – 7 – 8*

Tabla #25: Caminos básicos

Posteriormente al haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, 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 desea da para ver el funcionamiento del procedimiento.
- **Entrada:** Se muestran los parámetros que entran al procedimiento

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

Caso de prueba para el camino básico # 1. (1-2-3-4-5-8)

***Descripción*:** Se elimina un tipo de crédito existente en el sistema.

Condición de ejecución: Para poder eliminar un tipo de crédito este no puede estar siendo usado por ninguna solicitud.

Resultados esperados: Se espera que se elimine un nuevo tipo de crédito.

Resultados: En la Base de Datos DatTipocredito actualiza la propiedad de eliminado de cada tipo de crédito con la fecha actual, indicando que el crédito fue eliminado.

Caso de prueba para el camino básico # 2. (1-2-3-4-6-8)

***Descripción*:** Se elimina un tipo de crédito existente en el sistema.

Condición de ejecución: Para poder eliminar un tipo de crédito este no puede estar siendo usado por ninguna solicitud.

Resultados esperados: Se espera que se elimine un nuevo tipo de crédito.

***Resultados*:** En la Base de Datos DatTipocredito el crédito seleccionado no se encuentra, se manda un mensaje indicando que el crédito no existe.

Caso de prueba para el camino básico # 3. (1-2-7-8)

Descripción: Se elimina un tipo de crédito existente en el sistema.

Condición de ejecución: Para poder eliminar un tipo de crédito este no puede estar siendo usado por ninguna solicitud.

Resultados esperados: Se espera que se elimine un nuevo tipo de crédito.

***Resultados*:** En la Base de Datos DatTipocredito ya el crédito tiene la propiedad de eliminado con una fecha, se muestra un mensaje indicando que el tipo de crédito ya fue eliminado.

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

3.5. Integración entre componente

La arquitectura en 3 capas: presentación (view), negocio (controller) y acceso a datos (models), consiste en el flujo de los datos desde la vista hacia la capa de datos y viceversa, a través de los diferentes elementos que la componen. Consta de 4 nodos de integración: vista-controlador, controlador-modelo, modelo-framework Doctrine y Doctrine-Base de Datos. La comunicación entre las capas dentro de un mismo componente se realiza mediante llamadas a métodos o eventos de forma directa.

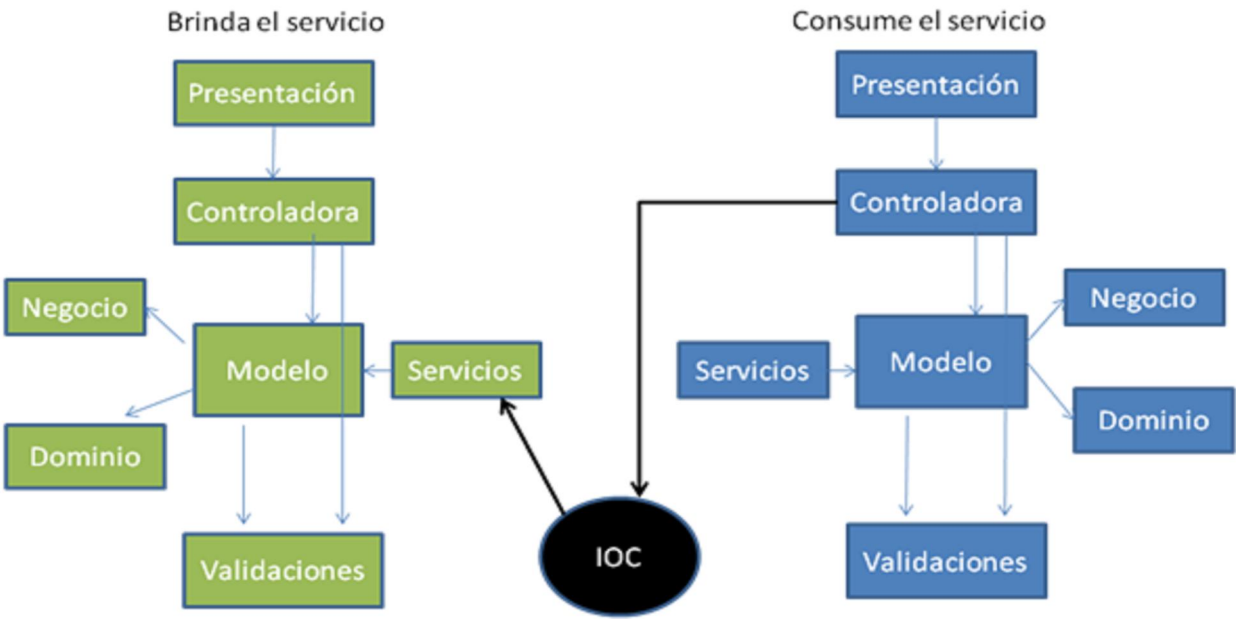


Figura #18: Integración entre componentes

Entre diferentes módulos y componentes, la integración se basa en el patrón Inversión de Control (IoC) y se realiza a través de un componente incluido en el framework Zend_Ext, que permite operar sobre distintos esquemas en la Base de Datos realizando las transacciones adecuadas. En dicho componente se define el fichero ioc.xml que contiene la ubicación de cada uno de los componentes y los servicios que ofrecen las clases services correspondientes. De esta manera, la puerta de entrada de cada componente es el paquete de las clases de servicios que buscan en los modelos o entidades las funcionalidades requeridas. Con la integración se persigue obtener una forma eficiente y flexible de combinar recursos internos o externos de los subsistemas.

Algunos de los servicios que presentaron problemas en el módulo de gestión de crédito se listan en la siguiente tabla.

ObtenerTipoSolicitudesPorId	Devuelve las solicitudes que coincidan con el id pasado por parámetro.
ObtenerSaldoEnBanco	Obtiene el saldo disponible en la cuenta bancaria de la entidad.
ObtenerOperacionesPorConceptoObjeto	Devuelve las operaciones en dependencia del concepto del objeto (Mora, Interés, Comisiones).
GenerarDerechoyContabilizar	Genera los derechos u obligaciones en el módulo Cobros y Pagos y posteriormente contabiliza.
InsertarNotificacion	Permite insertar una notificación del derecho generado.
FechaContable	Devuelve la fecha actual del servidor.

Tabla #26: Algunos servicios usados

3.6. Conclusiones

En el presente capitulo se garantizó la correcta implementación de la solución propuesta debido a que:

- Se describieron los estándares de codificación definidos por el centro CEIGE lo cual fue de gran utilidad en la implementación; sirvieron para aprender nuevos métodos y formas de organización del código, que permitirán mejorar la aplicación para futuros mantenimientos.
- Se aplicaron pruebas funcionales a la solución desarrollada, permitiendo encontrar no conformidades y dando la posibilidad de corregirlas a tiempo, obteniéndose un módulo que responda completamente a los requisitos funcionales existentes.
- Se aplicaron pruebas de Caja Blanca, verificando sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como en la falsa, obteniéndose resultados satisfactorios, que permitieron validar el correcto funcionamiento de las funcionalidades implementadas.

CONCLUSIONES

Con el desarrollo de los procesos Solicitud y Aprobación del módulo de Gestión de Créditos, se puede concluir que:

Se realizó un estudio del estado del arte, acerca de sistemas similares que existen, logrando un mayor entendimiento del funcionamiento de los mismos, permitiendo así obtener las principales ventajas que presentan, para usarlas en el desarrollo del módulo de Gestión de Créditos.

Se utilizaron para el correcto funcionamiento de este módulo las metodologías, herramientas y lenguajes acordes a las necesidades del país de migrar al software libre.

Se realizó un correcto diseño de la propuesta solución, donde se tuvo en cuenta el patrón de arquitectura MVC y varios patrones de diseño, que sirvieron de ayuda en el desarrollo del módulo, dándole pasó a la implementación.

Se validó mediante la aplicación de métricas que el diseño realizado para la propuesta de solución esta descrito de manera simple y con una calidad aceptable.

Se realizaron pruebas funcionales y pruebas unitarias, garantizando así la calidad del software y comprobando que el sistema cumple con las necesidades del cliente.

Luego del análisis anterior, se puede afirmar, que la presente investigación alcanzó los objetivos propuestos.

RECOMENDACIONES

Con el fin de contribuir a un mayor desarrollo de la presente investigación, se realizan las siguientes recomendaciones:

1. Realizar la liberación de la aplicación por el equipo de calidad de software de la Universidad.
2. Seguir desarrollando el Módulo de Gestión de Créditos.
3. Identificar nuevas funcionalidades del negocio a desarrollar en versiones posteriores del sistema.

REFERENCIAS

1. **Consultores**. [En línea] [Citado el: 2 de 10 de 2012.] http://www.jcbconsultores.com/.

2. **Global, SAP.** SAP Solutions. [En línea] 2010. [Citado el: 4 de 10 de 2012.] http://www.sap.com.

3. **Global , Website.** Sage. [En línea] http://www.sageerpx3.com/.

4. **Siscont.** Siscont. [En línea] [Citado el: 10 de 11 de 2012.] http://www.siscont.com/.

5. **Cabrera González, Lic. Miguel.** Sistema Económico Integrado VERSAT Sarasola. La Habana : s.n., 2004.

6. **Clamens, Szyperski.** Component Software – Beyond Object-Oriented Programming. 1997.

7. **Shaw, D. y Garlan, M.** Software Architecture: Perspectives. 1996.

8. **Martínez, Iribarne y Luis F.** Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. 2003.

9. **Centro de Soluciones de , Gestión.** Definición del ciclo de vida de los proyectos de desarrollo de software. La Habana : s.n., 2013.

10. **Systems, Popkin Software.** Modelado de Sistemas com UML. [Sección de] [Citado el: 20 de 11 de 2012.] http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf.

11. Visual Paradigm Company Overview. [En línea] [Citado el: 22 de 11 de 2012.] http://www.visual-paradigm.com/aboutus/.

12. **Torre, Aníbal de la.** Lenguajes del lado servidor o cliente. [En línea] 2006. [Citado el: 24 de 11 de 2012.] http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.

13. **Eguiluz, Javier.** Introducción a JavaScript.

14. **Función, Tu.** Tutorial Básico de Ajax. [En línea] 2006. [Citado el: 25 de 11 de 2012.] http://www.tufuncion.com/tutorial_basico_ajax.

15. **Van Der, Christian.** ¿Qué es el PHP? [En línea] [Citado el: 1 de 12 de 2012.] http://www.maestrosdelweb.com/editorial/phpintro/.

16. **Framework, Zend.** Introducción a Zend Framework. [En línea] 2010. [Citado el: 10 de 1 de 2013.] http://manual.zfdes.com/es/introduction.overview.html.

17. **Gómez, Pablo.** Instalación de Doctrine ORM en Debian/Ubuntu. [En línea] 2009. [Citado el: 10 de 1 de 2013.] http://www.arzion.com/empresa-de-internet/posts/Instalacion-de-Doctrine-ORM-en-DebianUbuntu.

18. **Corzo, Giancarlo.** ExtJs lo bueno, lo malo y lo feo. [En línea] 2008. [Citado el: 10 de 1 de 2013.] http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo.

19. **Briano, Fernando.** Control de versiones con Subversion. [Sección de] [Citado el: 2 de 2 de 2013.] http://picandocodigo.net/downloads/docs/subversion-presentacion-01.pdf.

20. **Función, Tu.** Los mejores IDEs para PHP. [En línea] 2007. [Citado el: 10 de 2 de 2013.] http://www.tufuncion.com/ide-php.

21. **CiberAula.** Una introducción a Apache. [En línea] 2006. [Citado el: 11 de 2 de 2013.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

22. **Cadavid López, Alejandro.** Mozilla Firefox, el navegador web del momento. [En línea] 2004. [Citado el: 12 de 2 de 2013.] http://www.maestrosdelweb.com/editorial/firefox/.

23. **ROSIQ, SERGIO EZEQUIER.** Base de Dato y su aplicación con SQL. s.l. : ISBN 987-526-213-7.

24. **UptoDown.** PostgreSQL. [En línea] 2009. [Citado el: 15 de 2 de 2013.] http://postgresql.uptodown.com/.

25. **Clements, Paul y Bass, Leen.** Software Architecture in practice. 1998.

26. **Gamma, Eric, y otros, y otros.** Design Patterns Elements of Reusable Object-Oriented Software.

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

28. **Aprendizaje, Entorno Virtual de.** Bienvenidos al Entorno Virtual de Aprendizaje. [En línea] [Citado el: 2 de 3 de 2013.] http://eva.uci.cu/mod/resource/view.php?inpopup=true&id=8505.

29. ¿Qué es un Patrón de Diseño? [En línea] [Citado el: 11 de 3 de 2013.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

30. **EcuRed.** Métricas para validar el diseño. [En línea] [Citado el: 2 de 4 de 2013.] http://www.ecured.cu/index.php/M%C3%A9trica_de_dise%C3%B1o.

31. **CEIGE, Centro.** Estándares de codificación. La Habana : s.n., 2013.

32. **Software, Pruebas de.** Gestión de Calidad y Pruebas de Software. [En línea] [Citado el: 22 de 4 de 2013.] http://pruebasdesoftware.com.

ANEXO 1

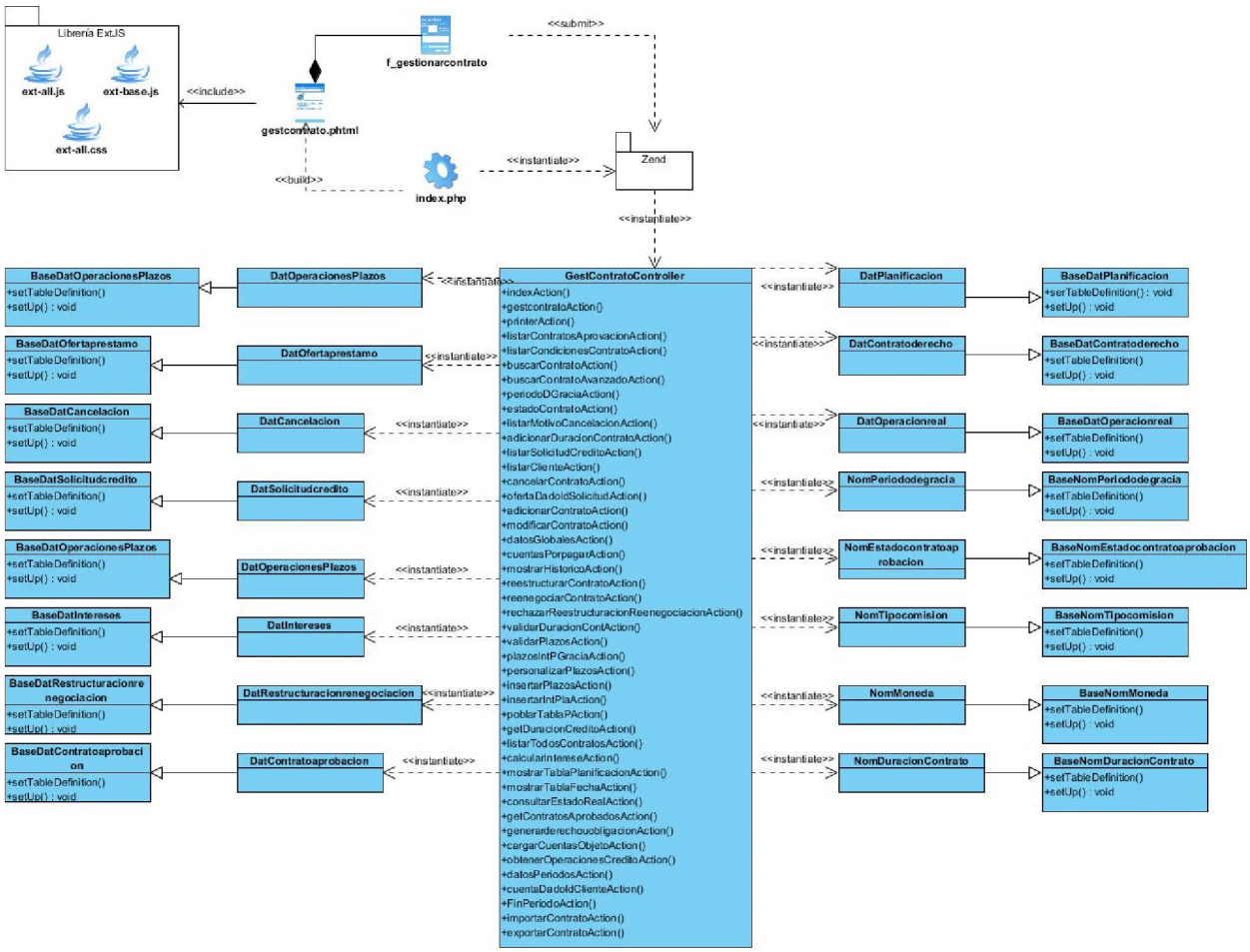


Figura #22: Diagrama de clases con estereotipos web del requisito gestionar contrato

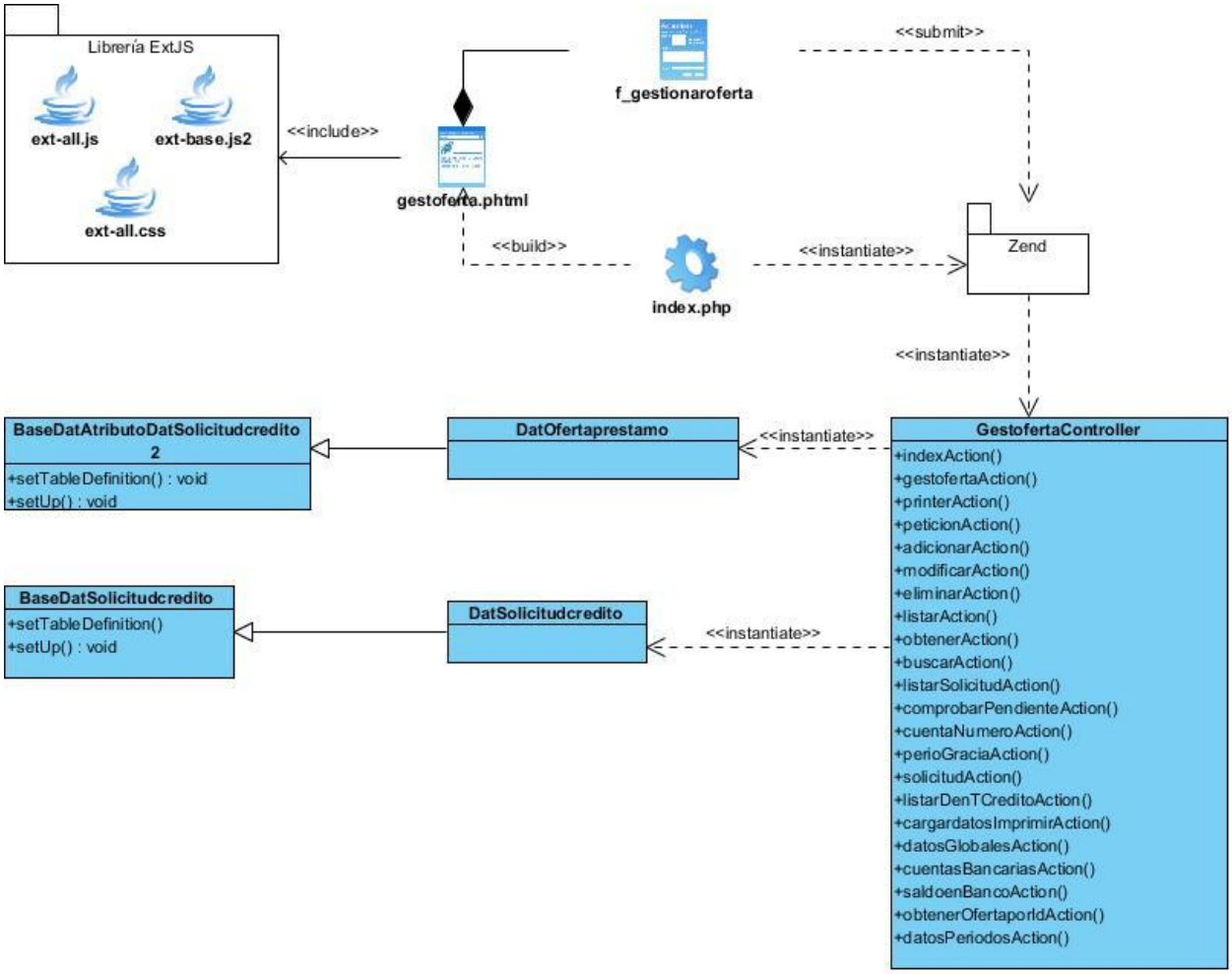


Figura #23: Diagrama de clases con estereotipos web del requisito gestionar oferta

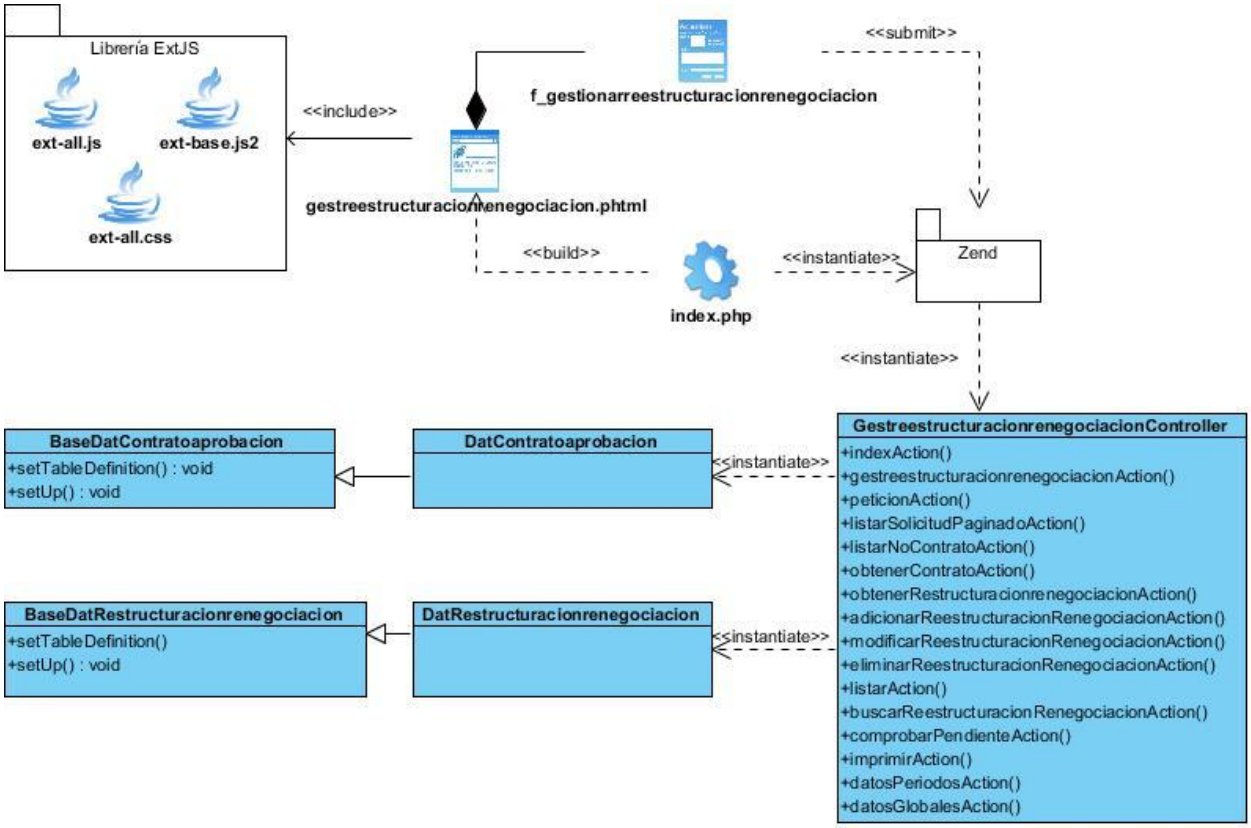


Figura #24: Diagrama de clases con estereotipos web del requisito gestionar restructuración o renegociación.

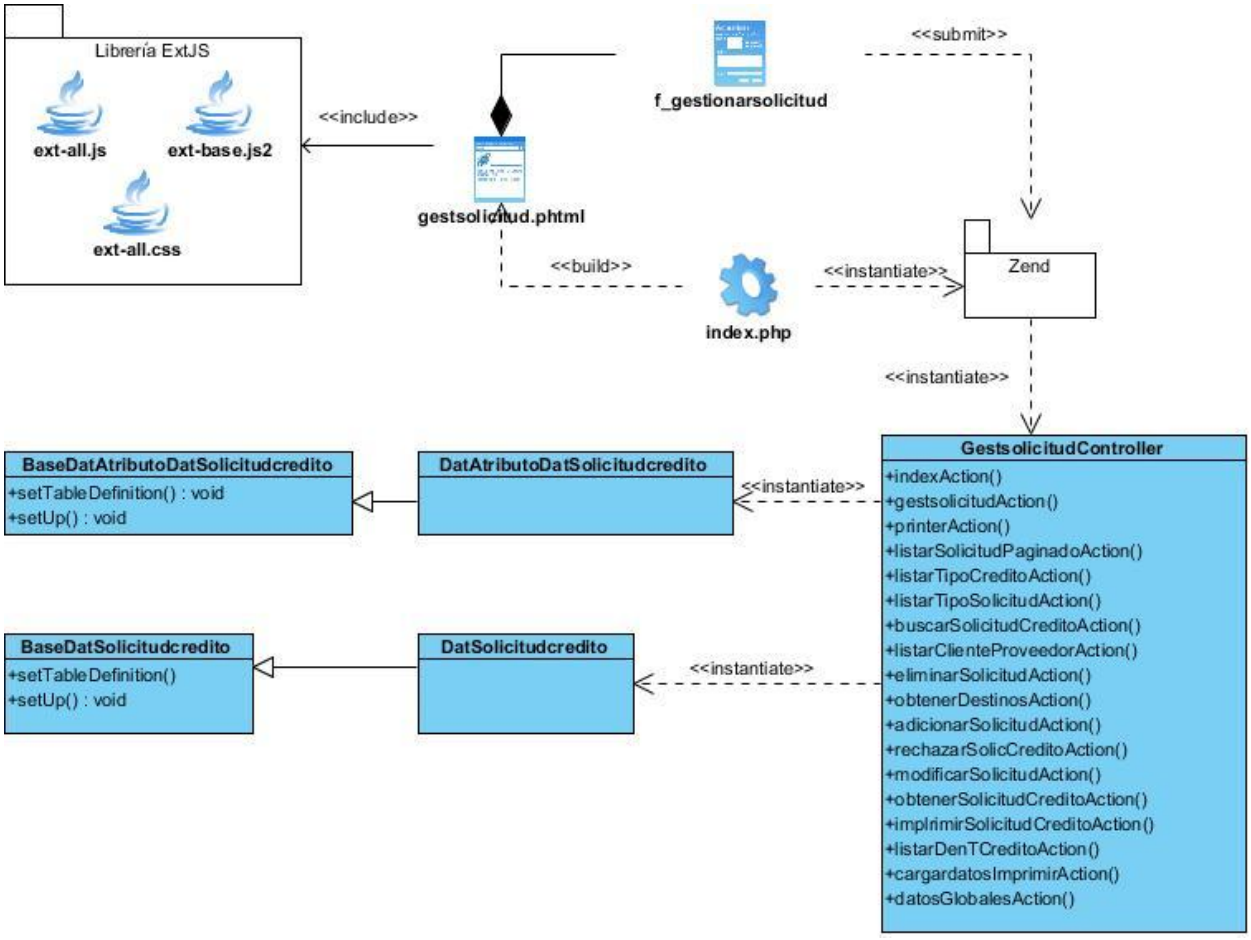


Figura #25: Diagrama de clases con estereotipos web del requisito gestionar solicitud

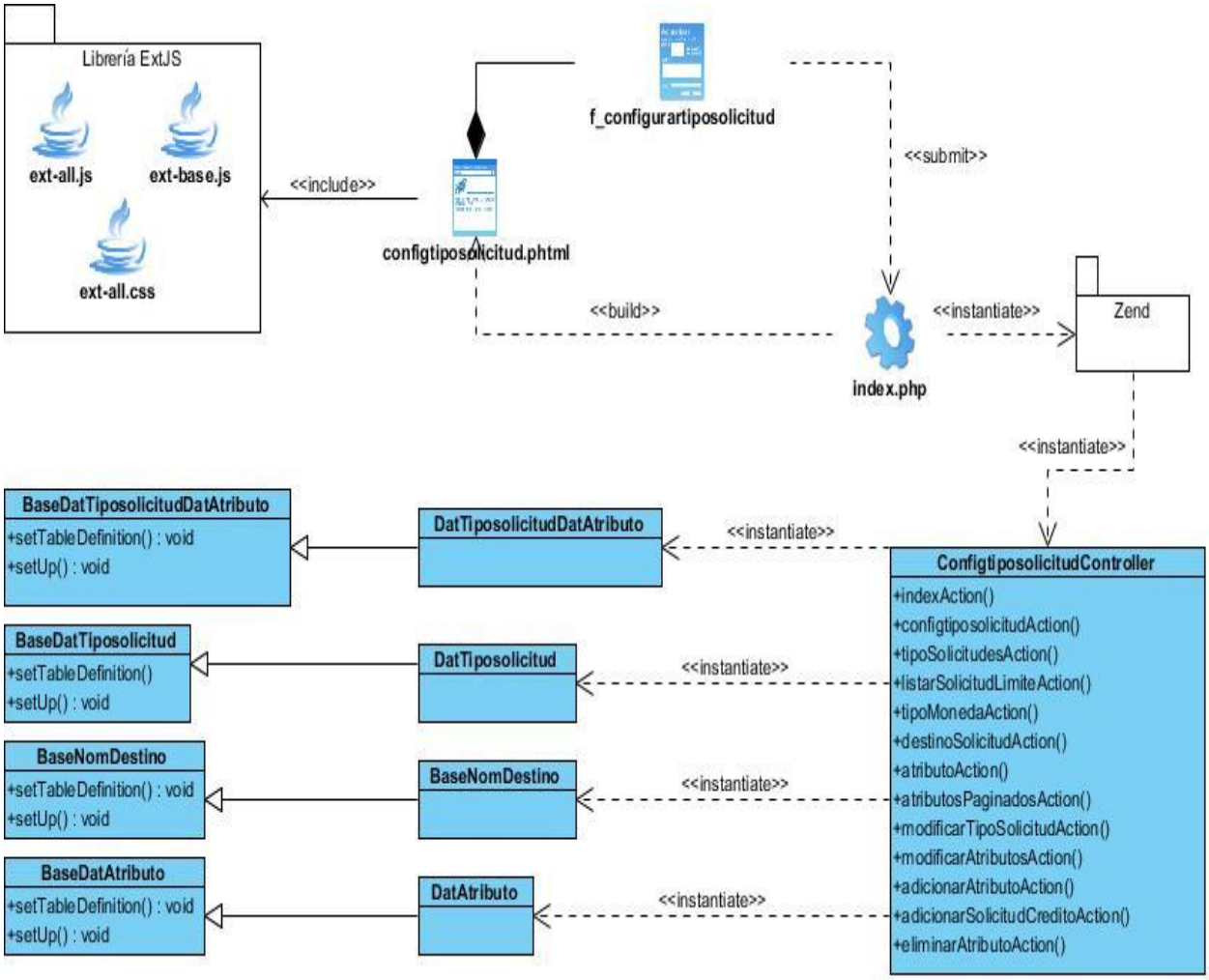


Figura #26: Diagrama de clases con estereotipos web del requisito configurar tipo de solicitud

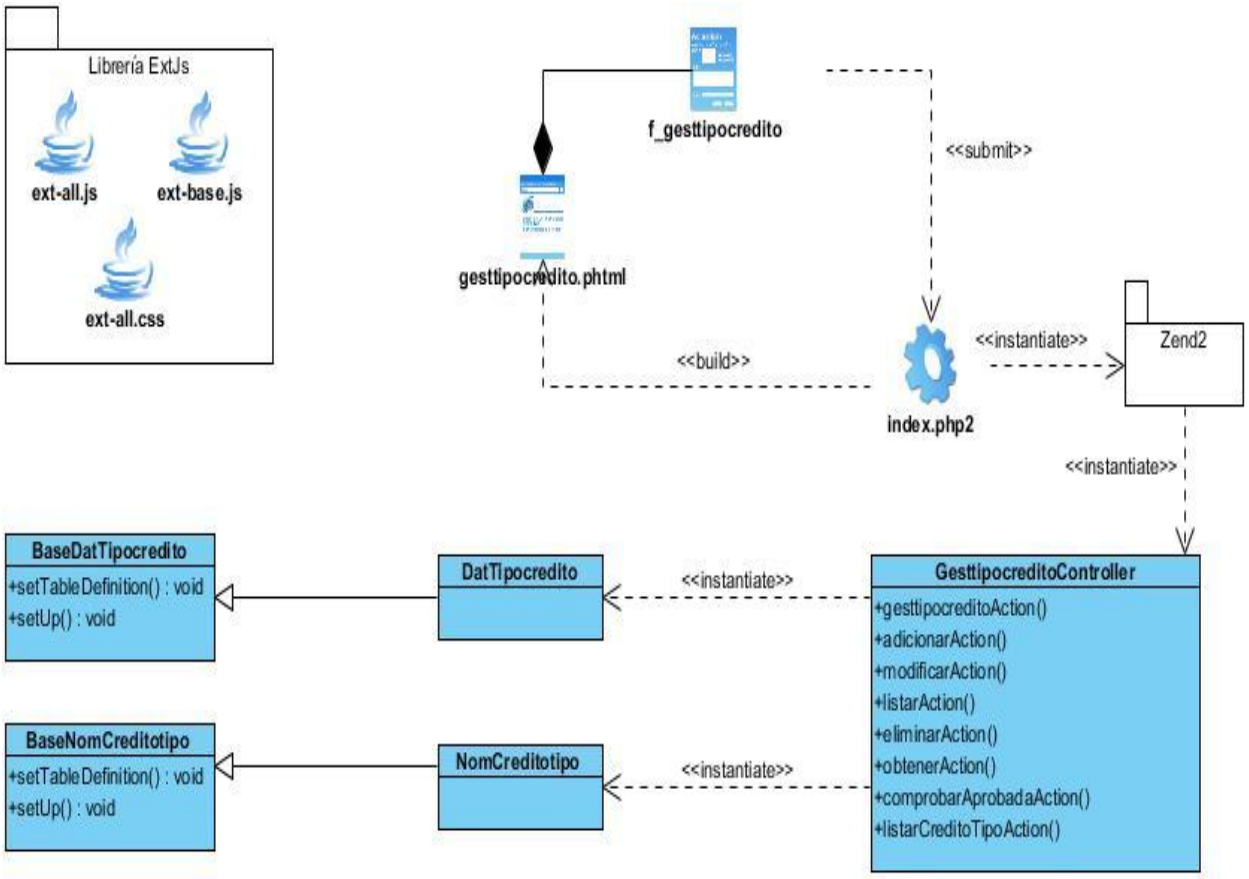


Figura #27: Diagrama de clases con estereotipos web del requisito gestionar tipo de crédito

ANEXO 2

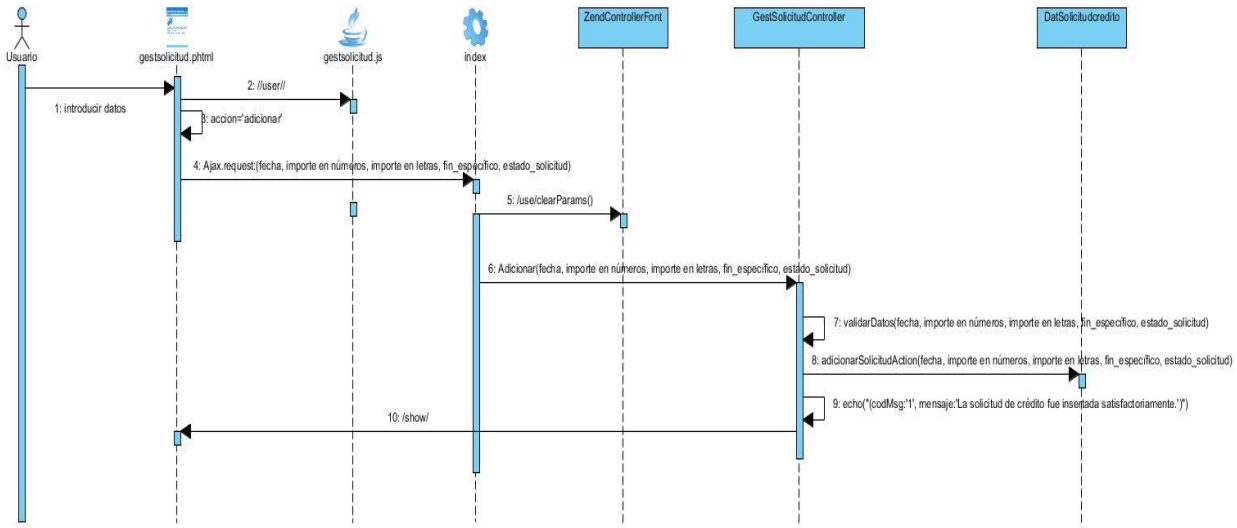


Figura #28: Diagrama de secuencia del requisito adicionar solicitud de crédito

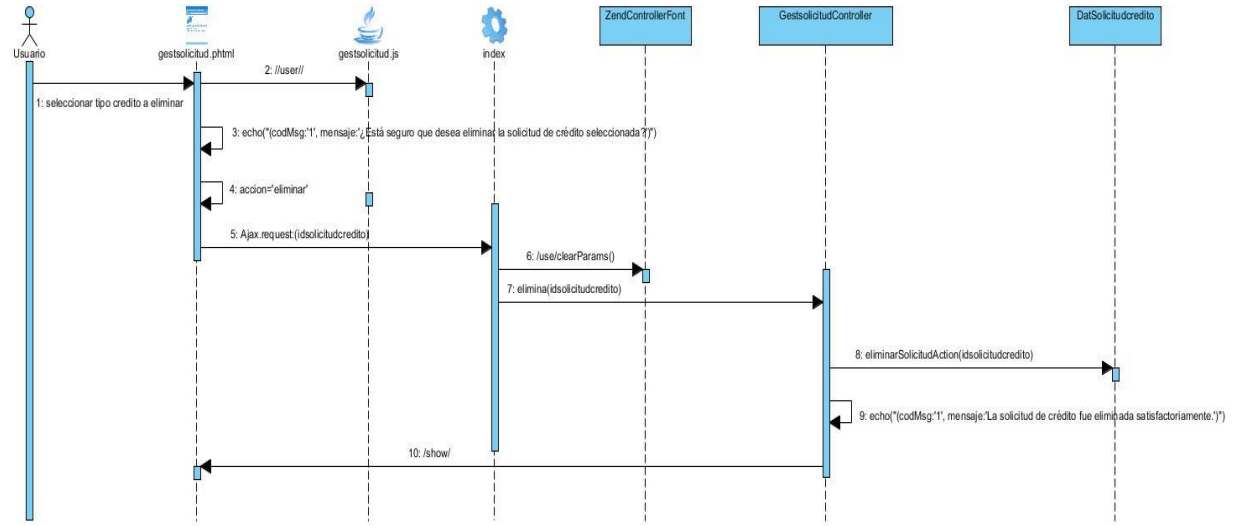


Figura #29: Diagrama de secuencia del requisito eliminar solicitud de crédito

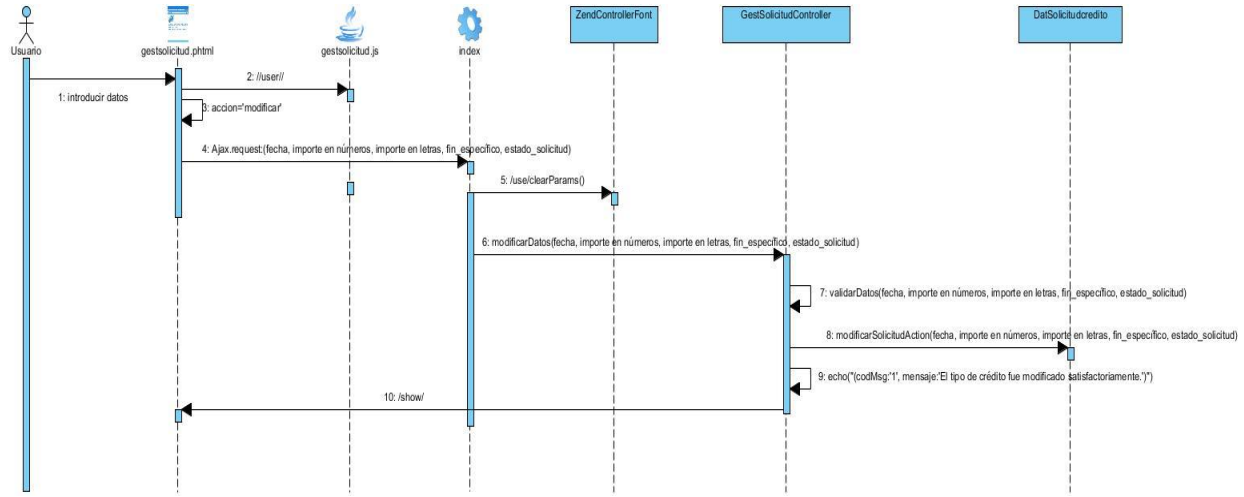


Figura #30: Diagrama de secuencia del requisito modificar solicitud de crédito

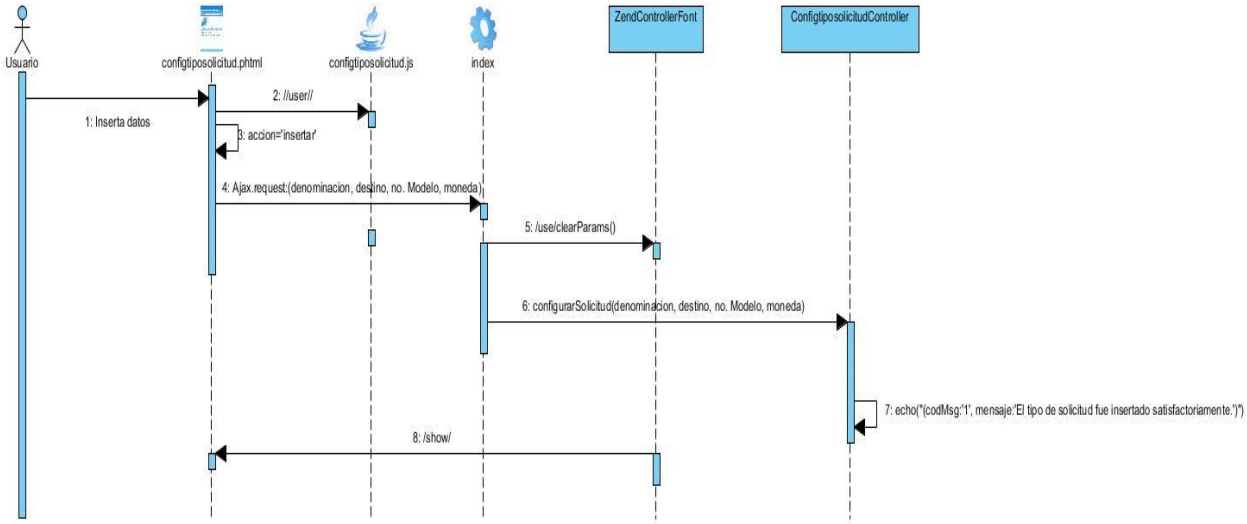


Figura #31: Diagrama de secuencia del requisito configurar solicitud de crédito

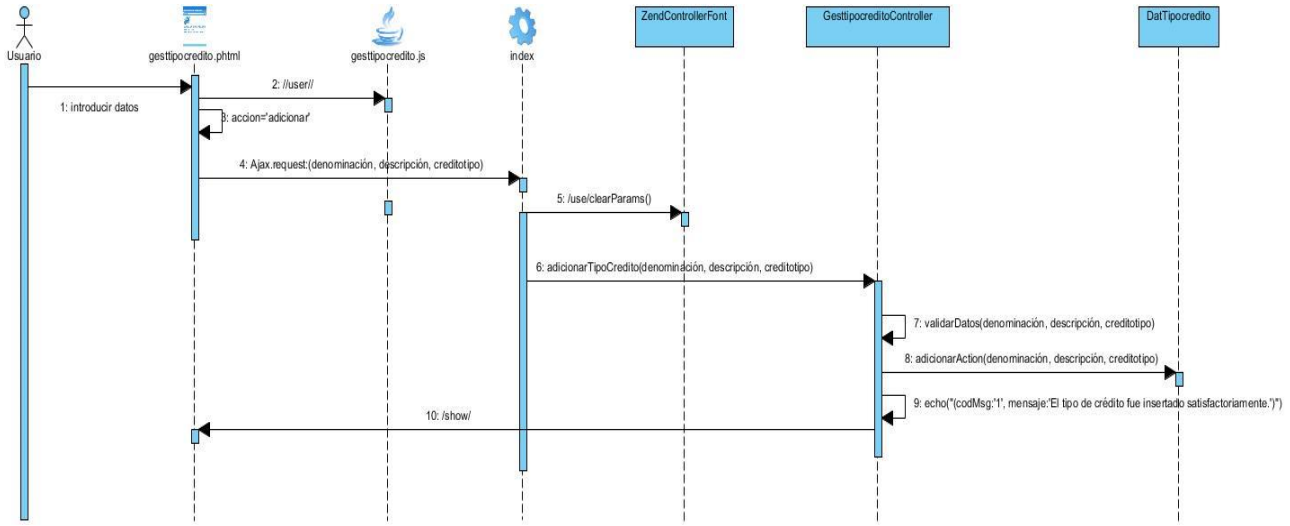


Figura #32: Diagrama de secuencia del requisito adicionar tipo de crédito

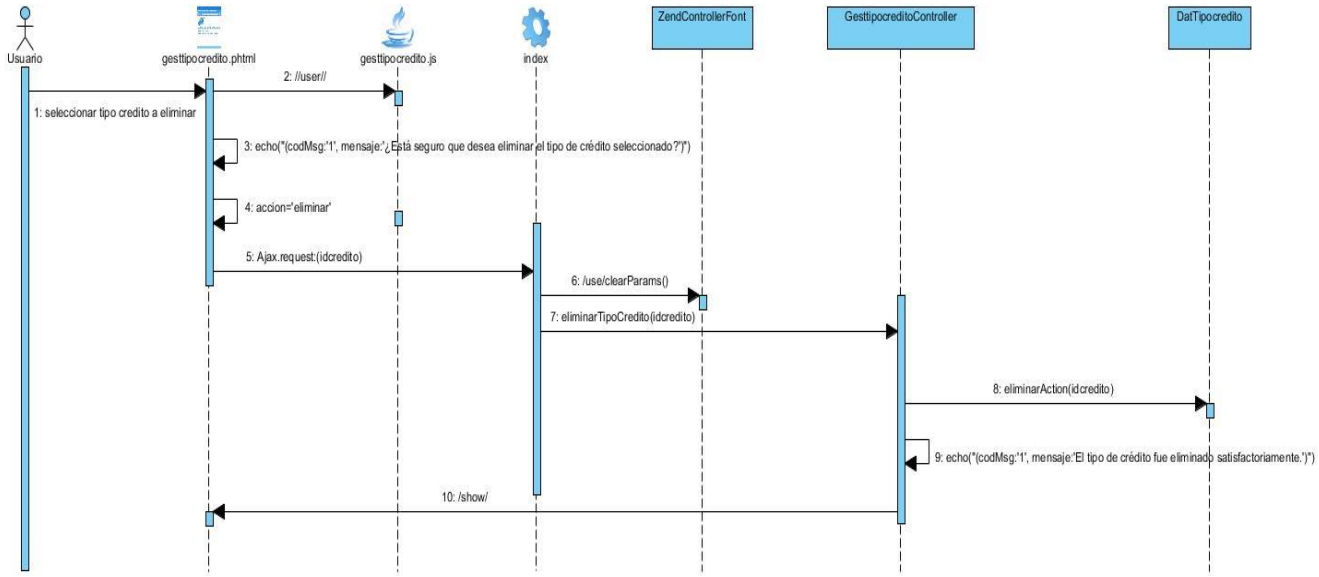


Figura #33: Diagrama de secuencia del requisito eliminar tipo de crédito

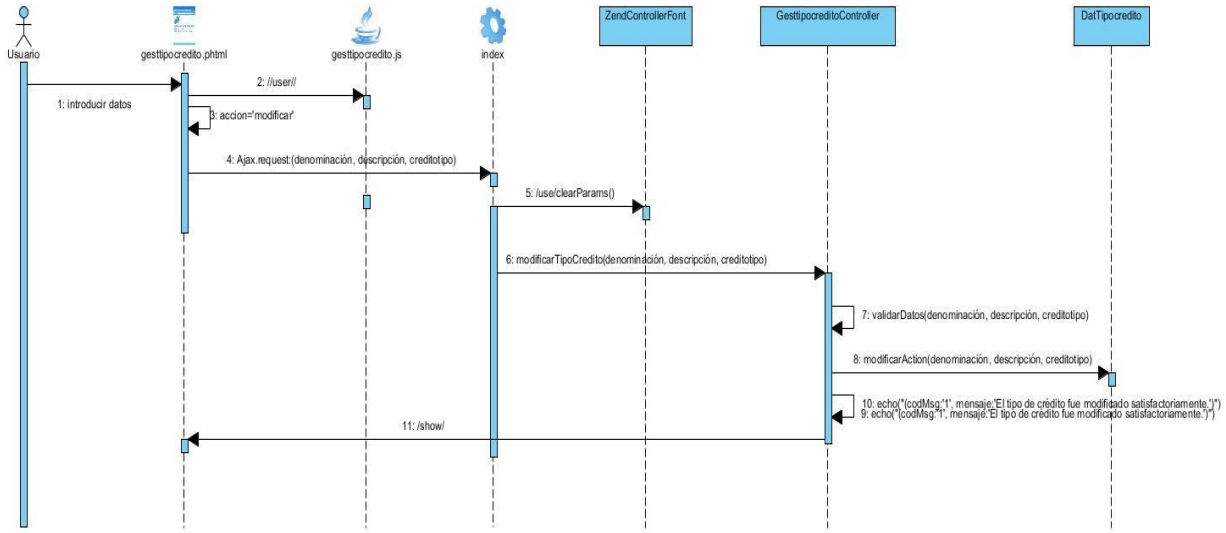


Figura #34: Diagrama de secuencia del requisito modificar tipo de crédito

ANEXO 3

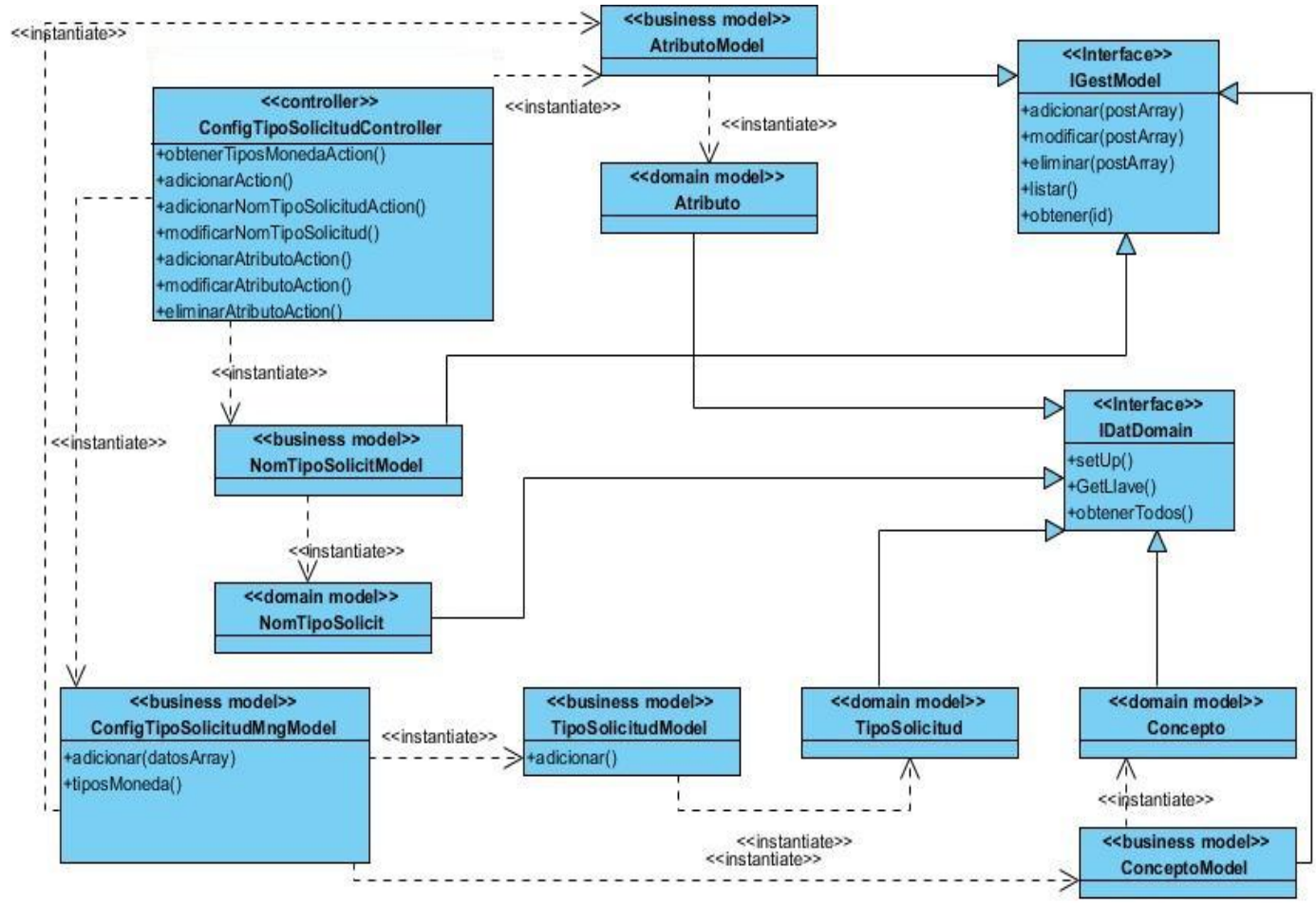


Figura #35: Clases del diseño del requisito configurar tipo de solicitud

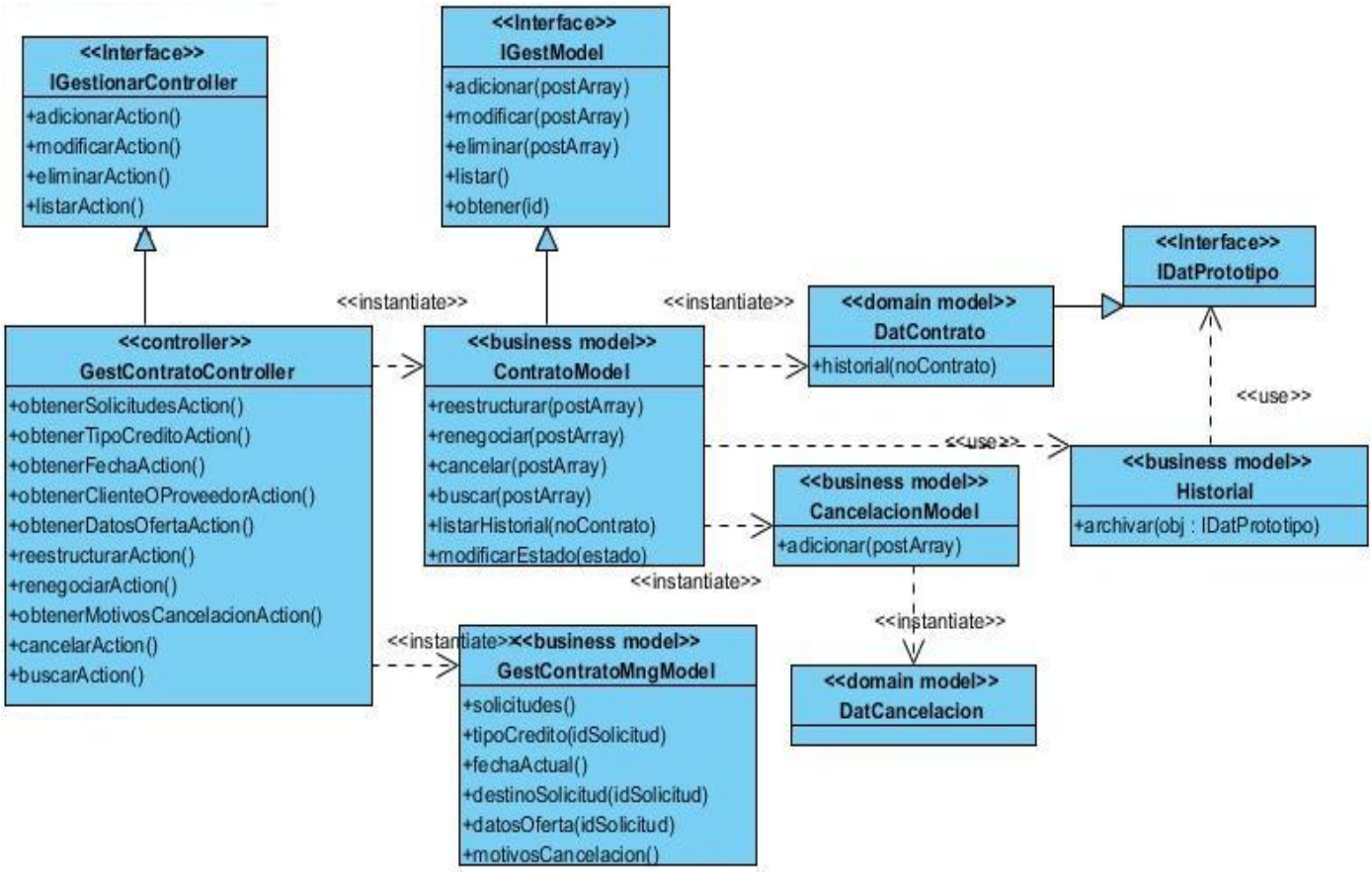


Figura #36: Clases del diseño del requisito gestionar contrato

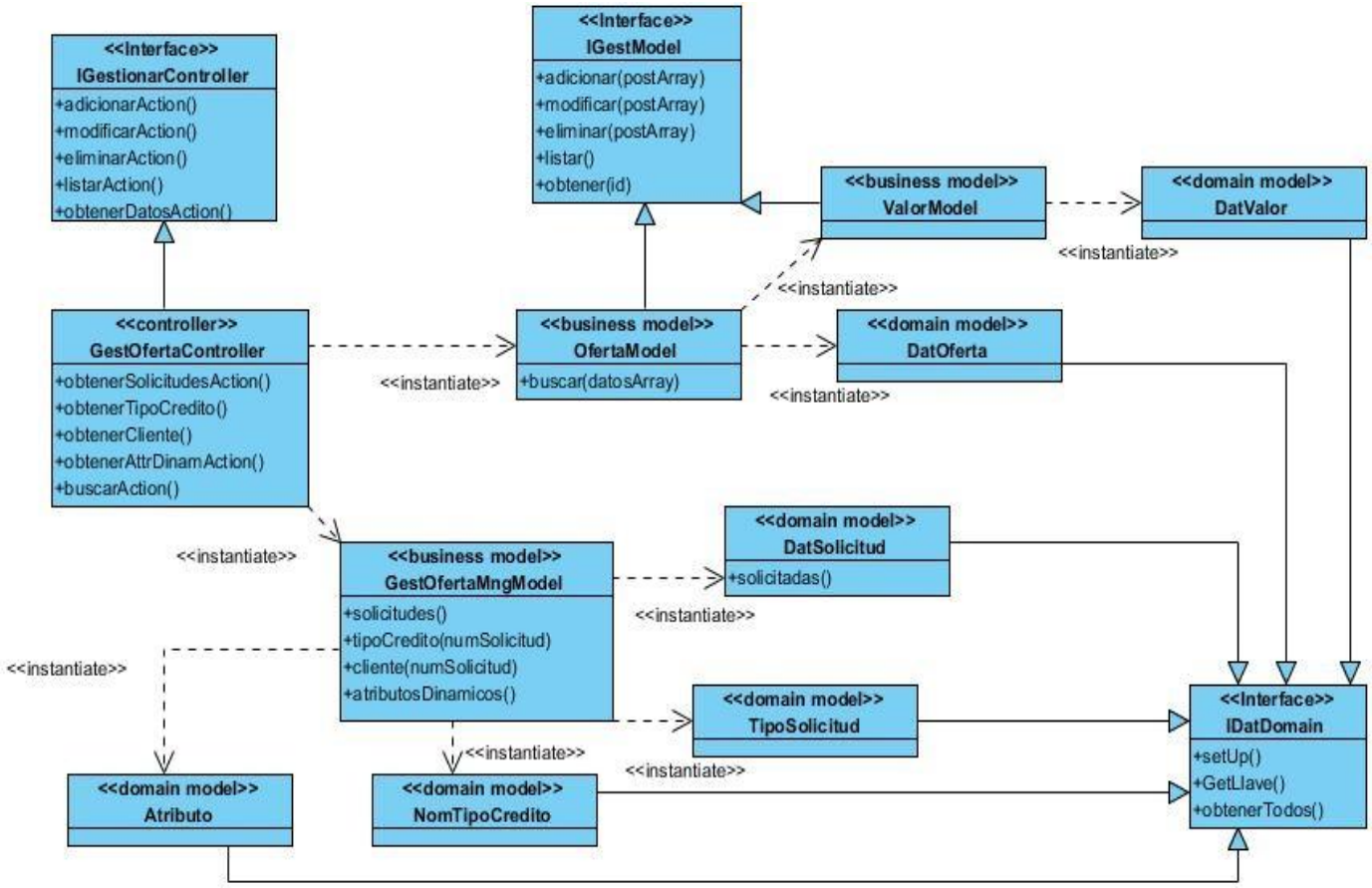


Figura #37: Clases del diseño del requisito gestionar oferta

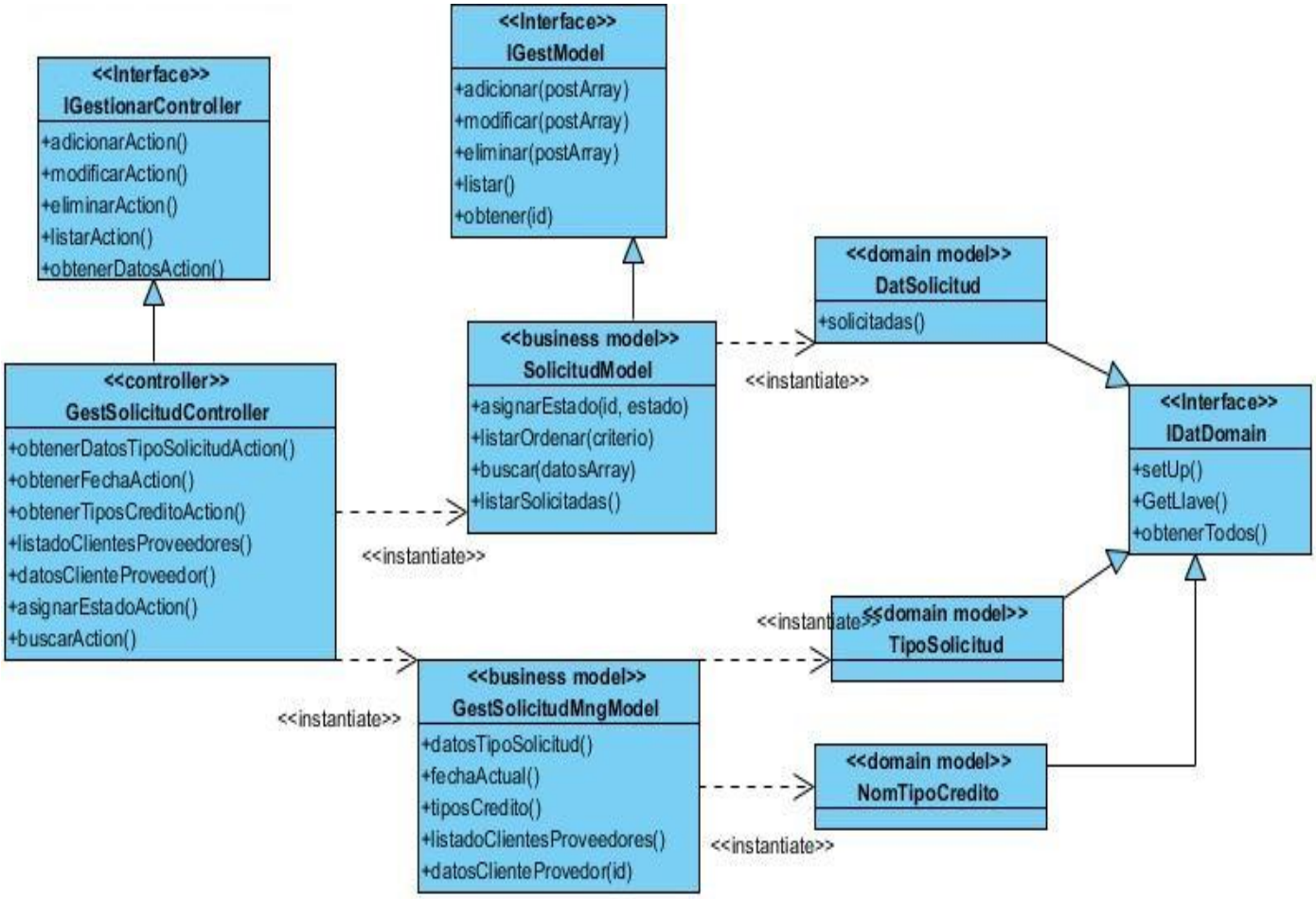


Figura #38: Clases del diseño del requisito gestionar Solicitud

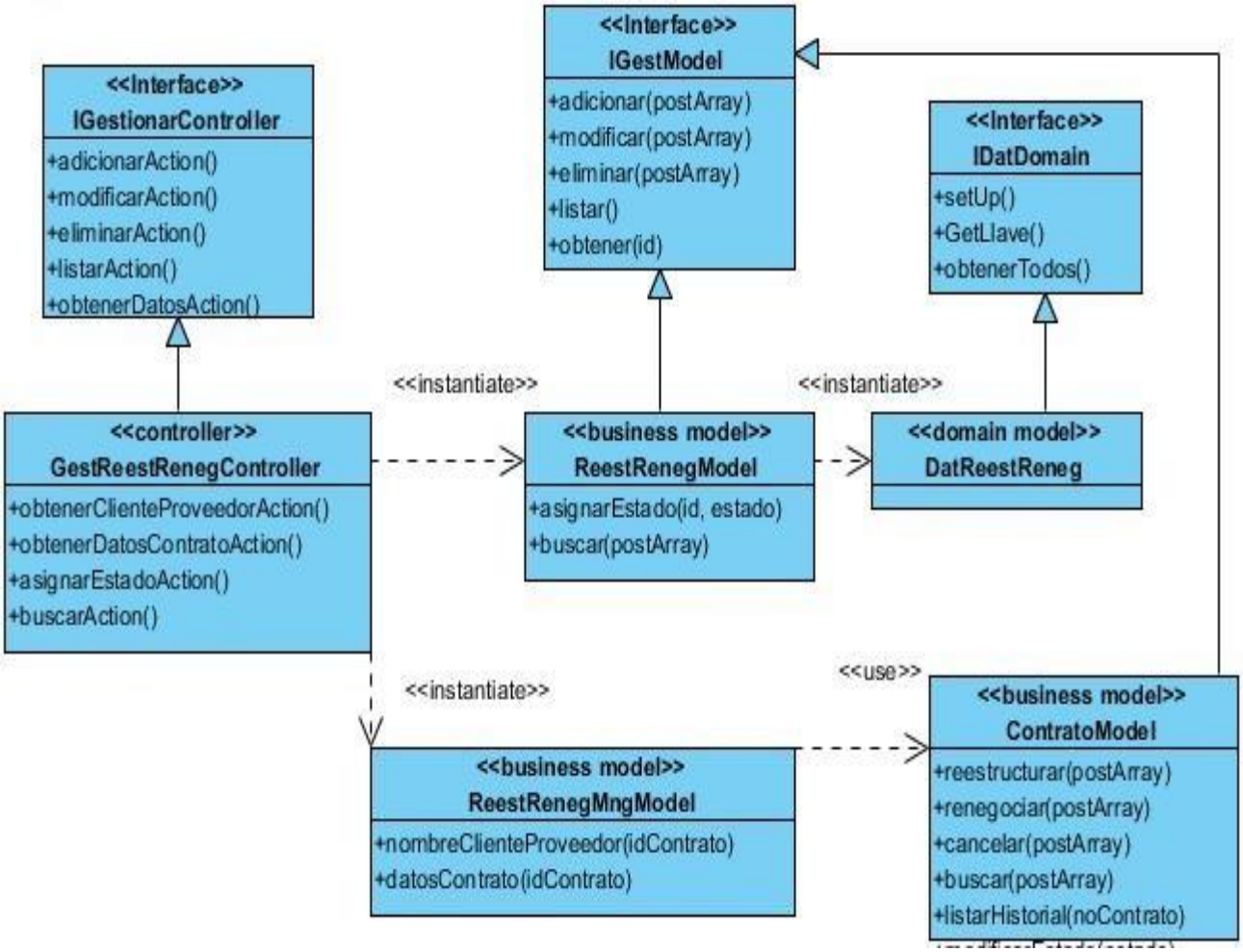


Figura #39: Clases del diseño del requisito gestionar restructuración o renegociación

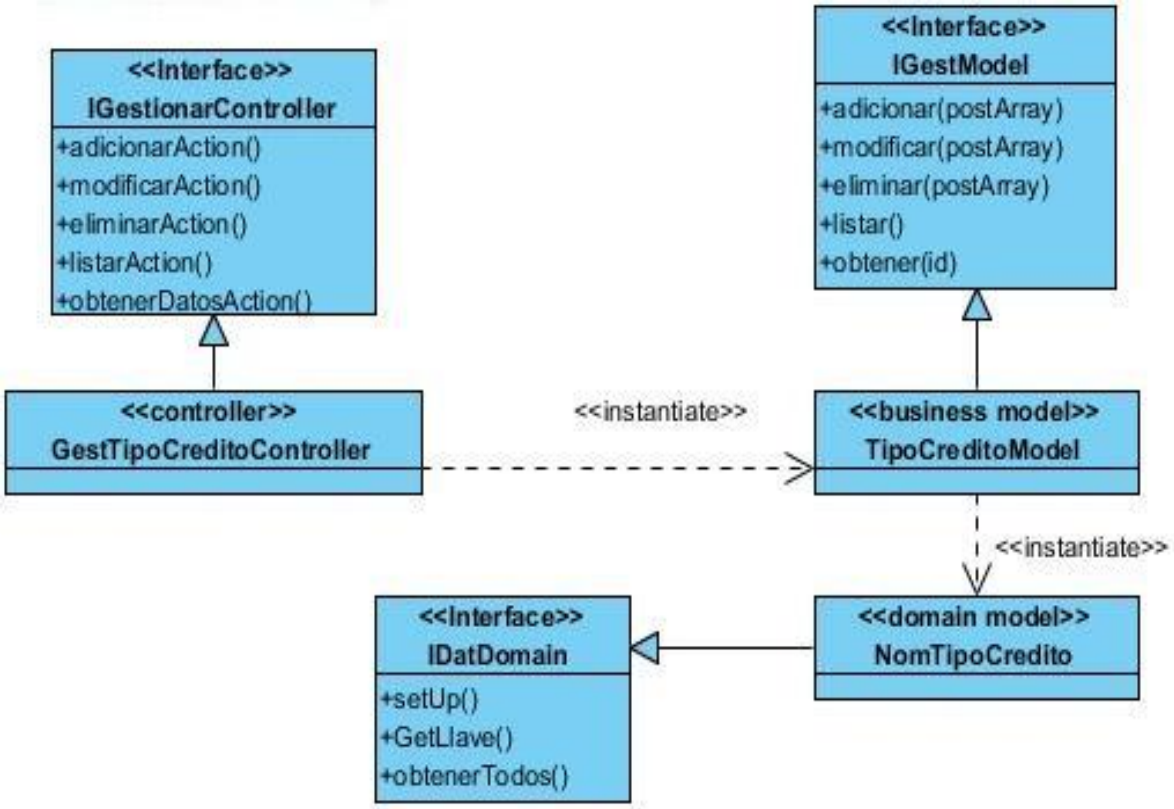


Figura #40: Clases del diseño del requisito gestionar tipo de crédito

ANEXO 4

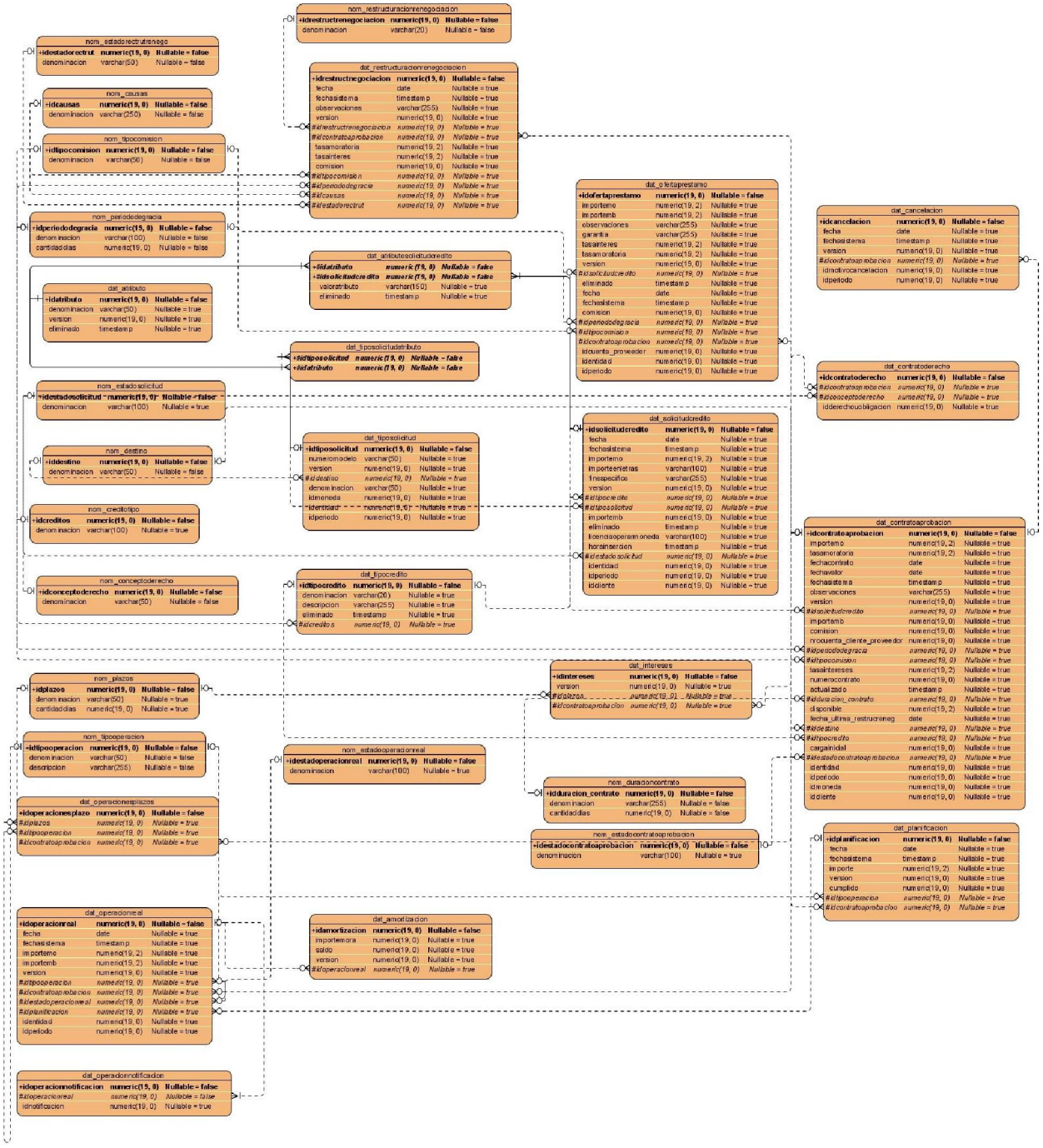


Figura #41: Diagrama entidad – relación del módulo Gestión de Créditos