Universidad de las Ciencias Informáticas

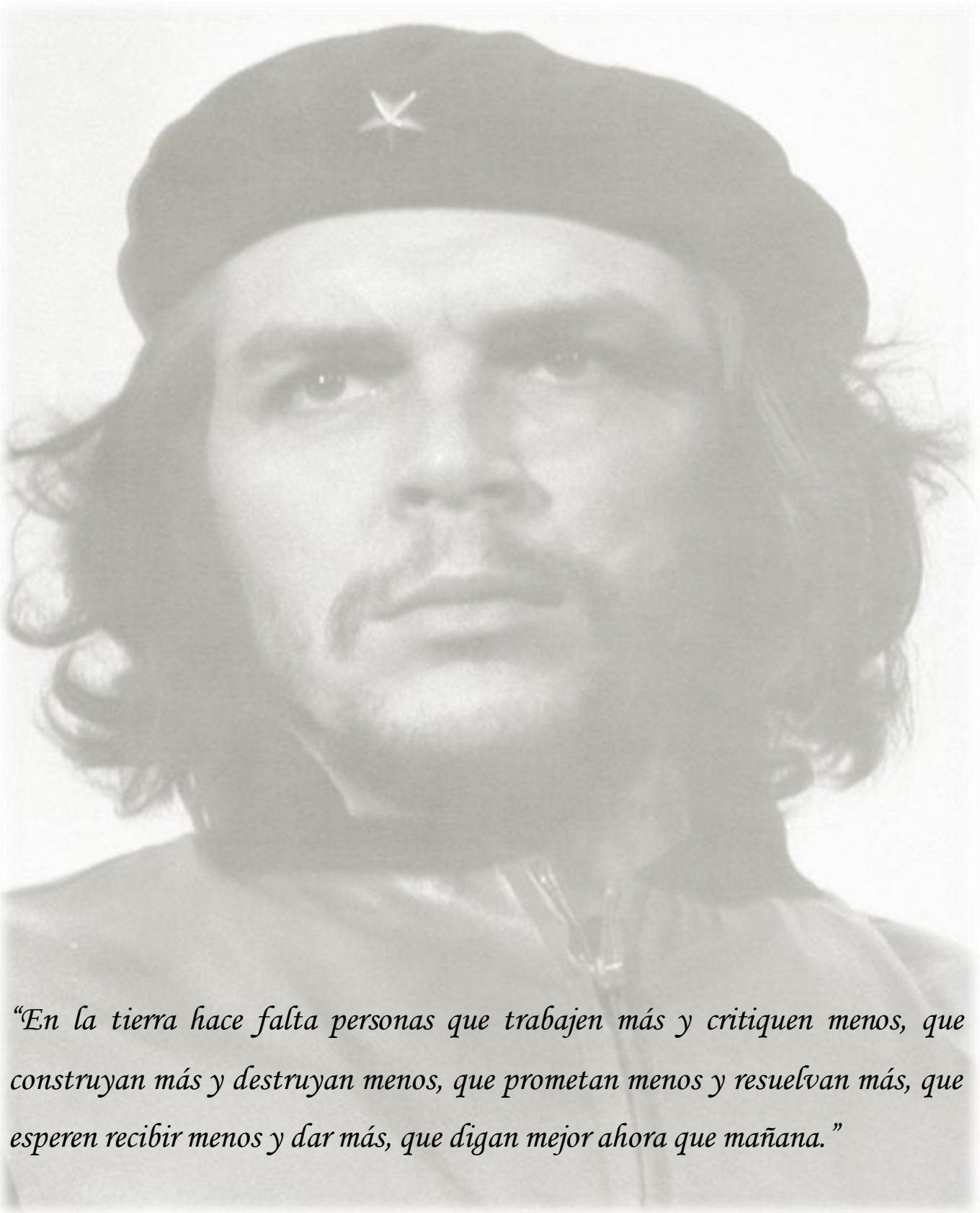
Facultad 3

Título: Diseño e implementación del módulo Análisis de Riesgos de Crédito de Quarxo.

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

Autor: Areanne Leyva Rivera
Tutor: Ing. Yoan Antonio López Rodríguez

La Habana, junio de 2012.
"Año 54 de la Revolución".



"En la tierra hace falta personas que trabajen más y critiquen menos, que construyan más y destruyan menos, que prometan menos y resuelvan más, que esperen recibir menos y dar más, que digan mejor ahora que mañana."

DECLARACIÓN DE AUTORÍA

Declaro que yo Areanne Leyva Rivera soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

```Autor: Areanne Leyva Rivera Tutor: Yoan A. López Rodríguez
```

DATOS DE CONTACTO

Ing. Yoan Antonio López Rodríguez.
Ingeniero en Ciencias Informáticas, graduado en Julio del 2008, Título de Oro, instructor. En la producción ejerce como Jefe de Equipo en el Proyecto SAGEB (Sistema Automatizado para la Gestión Bancaria) perteneciente al Dpto. de Soluciones Financieras del Centro CEIGE. Docentemente se desempeña como profesor de Teleinformática de la Facultad 3.

AGRADECIMIENTOS:

A mi mamá por apoyarme siempre y esforzarse como lo ha hecho hasta ahora para que yo tuviera una mejor preparación profesional. A mi abuela mami, por ser tan buena conmigo. A mi tía mima y mi tía Baby por haber estado ahí conmigo siempre que las necesité. A mis hermanos: Coco, Papo y Yailén por quererme tanto. A mi papá por apoyarme en mis decisiones y por su esfuerzo. A mis tíos, mis primos.

A mi familia en general por confiar siempre en mí.

A mi novio, por ser mi guía y mi sostén, por creer en mí, por quererme y apoyarme tanto como lo hizo estos dos últimos años.

A la familia de mi novio por acogerme como una más de la familia.

A mi padrastro y mi madrasta por su apoyo.

A Maidelis por comportarse como una hermana más.

A mis amistades de siempre Yaisel, Lisandra, Yuna, Yaritza y Yanisleidys.

A los integrantes del proyecto Banco, especialmente a Manuel y a Puig. También a Yakelín, Adrián, Choy, Dariel, Danis, Evelio, Leo y Vladimir.

Al tutor de mi novio, al cual admiro mucho, por haberme apoyado.

A los muchachos del grupo Dailiana, Lucrecia, Lilian, Yanier, Wilfredo, Nelson, Emilio, Marcel y el Bode.

A los profesores que durante toda la etapa de la universidad me prepararon para desempeñarme como una profesional.

A todos los que creyeron en mí.

DEDICATORIA

A mi mamá, mi abuela mami y mi tía mima.

A mi abuelo dondequiera que se encuentre.

A mi familia.

A mi novio.

A todos por su apoyo y comprensión en la realización de este sueño que es de ellos también.

RESUMEN

En estos momentos el Banco Nacional de Cuba (BNC, en lo adelante) en conjunto con la Universidad de las Ciencias Informáticas (UCI, por sus siglas en español) se encuentran inmersos en la tarea de desarrollar el sistema Quarxo, con el objetivo de informatizar un gran número de procesos que se llevan a cabo actualmente en esta entidad bancaria.

El presente trabajo de diploma tiene como objetivo desarrollar un módulo de análisis de riesgo de crédito para Quarxo, que agilice la entrega de los dictámenes de riesgo a la alta gerencia del banco, el cual pretende informatizar el proceso de otorgamiento de créditos en el BNC. El diseño fue realizado sobre la base de patrones que facilitan y garantizan el desarrollo ágil y la calidad del sistema. Para la implementación de dicho módulo se utilizaron herramientas y tecnologías de la plataforma Java, con el objetivo de aprovechar su alto potencial para el desarrollo de aplicaciones web, además de ser software libre y multiplataforma. Se hizo uso de algunos elementos fundamentales, tales como Spring Web Flow en la parte de implementación, para facilitar el control de los flujos web.

Con el objetivo de obtener una solución que cumpla con las expectativas del cliente, se llevó a cabo la validación del diseño de la solución propuesta a través de métricas. Por otra parte, para la implementación se realizaron pruebas de caja blanca y pruebas de caja negra, las cuales arrojaron resultados satisfactorios.

Palabras clave: Riesgo, crédito, diseño, implementación.

TABLA DE CONTENIDO

INTRODUCCIÓN

Hoy en día el uso de las Tecnologías de la Información y las Comunicaciones (TIC, por sus siglas en español) es un factor muy importante para lograr mayor productividad y competitividad en el mercado global; las mismas juegan un papel estratégico en la informatización de los procesos de gestión llevados a cabo en las empresas.

Los bancos como entidades financieras no están ajenos a estos cambios. Ellos por su parte poseen el control económico del mundo y su actuación estratégica se ha visto favorecida en gran medida por la utilización de sistemas informáticos, como pilar fundamental para el incremento de la eficiencia en la toma de decisiones y en el resto de sus operaciones. En los bancos, una de las principales actividades que generan ingresos la representan, las operaciones activas a través de los créditos otorgados a los solicitantes, sin embargo este tipo de operaciones cuentan con un factor de riesgo, conocido como riesgo de crédito. Existen otros riesgos bancarios entre ellos: el riesgo de liquidez y el riesgo de mercado. El presente trabajo de diploma se centra en el riesgo de crédito, que se define básicamente como, la posibilidad de que el cliente no cumpla con los términos acordados en los contratos del crédito.

Un análisis inadecuado de la capacidad de pago de un solicitante, puede resultar en una mala calidad del préstamo, pérdidas de crédito sustanciales y el desgaste de las ganancias y el capital del banco. (1)

A raíz de los Lineamientos de la Política Económica y Social del Partido y la Revolución es necesario, como lo indica el lineamiento número cincuenta y uno: Establecer los mecanismos y condiciones imprescindibles que garanticen la agilidad en el otorgamiento de créditos y la recuperación de los mismos. (2)

En estos momentos en la UCI, se desarrolla un software para la gestión de los procesos en el BNC, el cual se denomina Quarxo. Como parte del desarrollo de este sistema se pretende integrar un módulo para el análisis de riesgo de crédito a clientes.

Actualmente en el BNC es necesario registrar de forma manual la información de los estados financieros de las empresas y a partir de la experiencia y los conocimientos de los analistas, se determina si es factible o no el otorgamiento del crédito. Además, en el BNC no es posible contar con un historial informatizado que permita obtener de una manera rápida y tipificada el comportamiento de sus deudores. Por otro lado, en la valoración de los dictámenes finales no se tienen en cuenta

todos los indicadores contables definidos para ello, pues un analista de riesgo debe analizar para cada cliente treinta indicadores contables, de los cuales analiza veinticinco. Recientemente, producto a los cambios en el desarrollo de la economía del país han ido aumentando las solicitudes de créditos, lo que implica una ardua labor en el área de análisis de riesgo de créditos. Por tales motivos, el proceso de determinación de factibilidad del otorgamiento del crédito se realiza con determinada lentitud, lo que trae consigo que se entreguen los dictámenes finales a la alta gerencia fuera del tiempo establecido.

Dada la situación problemática anteriormente planteada, se identificó el siguiente **problema a resolver**: Las deficiencias en el proceso de análisis de riesgo de crédito en el BNC, provocan retrasos en la entrega de los dictámenes de riesgo a la alta gerencia del banco.

El **objeto de estudio** del presente trabajo se enfocó hacia el proceso de análisis de riesgo de crédito en las entidades bancarias y el **campo de acción** hacia el proceso de análisis de riesgo de crédito mediante sistemas informáticos.

Para resolver el problema anteriormente planteado, se trazó como **objetivo general:** Desarrollar un módulo para el análisis de riesgo de crédito para Quarxo, que agilice la entrega de los dictámenes de riesgo a la alta gerencia del banco.

Objetivos específicos:

1. Realizar un estudio del estado del arte de los sistemas informáticos de gestión bancaria.
2. Diseñar el módulo Análisis de riesgo de crédito de Quarxo.
3. Implementar el módulo Análisis de riesgo de crédito de Quarxo.
4. Validar el módulo implementado.

Para cumplir lo planteado, se llevarán a cabo las siguientes **tareas**:

- Investigar sobre los procesos de análisis de riesgo de crédito mediante sistemas informáticos en el mundo.
- Analizar detalladamente la tecnología y la arquitectura definidas en el proyecto.
- Analizar los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
- Analizar los requisitos especificados para el desarrollo del módulo propuesto.
- Realizar el diseño del módulo.
- Implementar la solución.
- Realizar pruebas al módulo.

Idea a defender: Con el desarrollo de un módulo de análisis de riesgo de crédito para el sistema de gestión bancaria Quarxo, se agilizará la entrega de los dictámenes de riesgo a la alta gerencia en el BNC.

Posibles resultados:

- Un módulo de análisis de riesgo de crédito para el sistema bancario Quarxo.
- Disminución del tiempo de entrega de los dictámenes de riesgo a la alta gerencia en el BNC.

Para lograr la comprensión y claridad de los contenidos de la investigación realizada se ha estructurado el documento de la siguiente manera.

Capítulo 1: Fundamentación teórica. En este capítulo se realiza un estudio del sistema bancario cubano, los procesos relacionados con el otorgamiento de créditos, así como de varios sistemas informatizados para la gestión bancaria a nivel mundial que realizan este proceso. Además, se abordan temas de vital importancia a la hora de realizar el diseño y la implementación del módulo Análisis de riesgo de crédito de Quarxo, señalando lo referente al estudio de la metodología, las herramientas y las tecnologías, que guiarán el proceso del desarrollo de software.

Capítulo 2: Diseño y validación de la solución propuesta. Inicialmente se elabora el diseño de la propuesta de solución a partir de la especificación de requerimientos del módulo Análisis de riesgo de crédito, enfocado a la construcción de diagramas de clases de diseño, diagramas de paquetes y diagramas de secuencias, para conformar el modelo de diseño. Posteriormente se lleva a cabo la validación del diseño a partir de las métricas Tamaño operacional de clases y Tamaño de clases.

Capítulo 3: Implementación y pruebas de la solución propuesta. Se lleva a cabo la implementación del módulo, donde se explican los estándares de codificación utilizados, las clases principales y el uso de Spring Web Flow. Se obtienen además, los artefactos generados en esta etapa del software. Luego se valida la implementación a partir de la realización de las pruebas de unidad, brindando una explicación de las pruebas de caja blanca que fueron realizadas al código del software y las pruebas de caja negra que se realizaron a la interfaz del mismo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se abordan diferentes temas que constituyen la base para efectuar el diseño y la implementación del módulo Análisis de riesgo de crédito del sistema Quarxo. Se estudian varios modelos y sistemas informatizados que llevan a cabo el análisis de riesgo de crédito en el mundo. Además, se analiza la metodología, el uso de patrones de diseño, las herramientas y las tecnologías definidas para el desarrollo de la solución.

1.2 Sistema Bancario

Las nuevas exigencias de la economía interna y el desarrollo de las finanzas a escala internacional exigen un proceso de continuo perfeccionamiento del Sistema Bancario, garantizando su mayor modernización y competitividad. La importancia de la banca es primordial para el desarrollo de la economía, ya que su principal función es suministrar fondos a empresas públicas, privadas y personas naturales que los necesitan para poder cumplir con los compromisos de pagos contraídos con los proveedores, bienes y servicios.

Para desarrollar un sistema capaz de informatizar las actividades contables y el resto de las operaciones de un banco, es preciso conocer a fondo su funcionamiento. Inicialmente el banco tiene la función de captar pasivos a clientes, los cuales de una manera bien planificada se utilizan en la adquisición de intereses para la entidad. De la misma manera se realizan un grupo de operaciones tales como préstamos, transferencias y una serie de pagos. Además de las muchas negociaciones, ya sea entre clientes cercanos o a disímiles distancias que también son apoyadas por los bancos.

Cada banco se estructura de la manera más conveniente, contando con una serie de departamentos, en los cuales lleva a cabo las operaciones de una manera coordinada. (3)

1.2.1 Sistema Bancario Cubano

En la actualidad Cuba se encuentra en pleno proceso de transformación económica, con el fin de insertarse en los mercados internacionales y sentar las bases para un futuro desarrollo sostenido de la economía del país. Uno de los aspectos claves para lograr este fin es la elevación de la eficiencia empresarial, para lo cual es importante, no sólo los cambios que deben producirse en la gestión de las empresas, sino también el perfeccionamiento de su entorno financiero, imprescindible para alcanzar los objetivos que se persiguen. (3)

Entre los bancos cubanos se puede citar el BNC, el cual una vez liberado de las funciones de banca central y de rector del sistema bancario, continúa existiendo con el carácter de banco comercial, autorizado a ejercer funciones inherentes a la banca universal, teniendo además la función de registro, control, servicio y atención de la deuda externa que el Estado y el propio banco han contraído con acreedores extranjeros con la garantía del Estado. (4)

El uso de los sistemas informáticos resulta común en las entidades cubanas, orientados fundamentalmente a obtener una mayor eficiencia en la gestión empresarial. La UCI en conjunto con el BNC se encuentra desarrollando el sistema bancario Quarxo, por lo que la dirección del banco ha solicitado integrar al sistema Quarxo el proceso de análisis de riesgo de crédito. Con este fin se propone, desarrollar un módulo que permita llevar a cabo el análisis de riesgo de crédito; que tomará por entradas los estados financieros de una empresa o entidad jurídica y través de fórmulas dadas por los ratios financieros, indicará si la transacción resulta factible o no.

1.2.2 Procesos Bancarios

La banca requiere la aplicación eficaz de los más modernos procedimientos de planeamiento y conducción que puedan aportar sugerencias válidas e inmediatas a su rentabilidad y desarrollo. En un mercado tan competitivo como el bancario, la fuerza que impulsa a mejorar los procesos proviene de la necesidad de atraer, atender y satisfacer mejor a los clientes externos. En otras palabras, la mejora ha de ser una actividad continua, si el banco no quiere perder contacto con las necesidades actuales y futuras de su mercado objetivo.

Los procesos bancarios se pueden comprender a partir del estudio de los objetivos generales y específicos que tiene un banco como entidad financiera. La atención personalizada a los clientes y la búsqueda incesante de ingresos, resultan algunos de ellos, además de promover su actividad como instrumento, para el desarrollo del Comercio Exterior y sostener las relaciones con otros bancos. (3)

Las operaciones bancarias son todas aquellas operaciones de crédito practicadas por un banco de manera profesional. Las mismas se clasifican en activas, pasivas y neutras. Las operaciones activas son aquellas en las que el banco otorga el crédito, como ejemplos de ellas se tienen los préstamos y los descuentos. Las operaciones pasivas son aquellas en las que el banco recibe dinero de clientes y paga intereses por esas prestaciones, como ejemplos se tienen las Cuentas Corrientes, las de Ahorro, a Plazo Fijo y Cédulas Hipotecarias. Por su parte las operaciones neutras o accesorias son aquellas donde el banco no recibe ni otorga crédito, como las operaciones de mediación, donde el

banco solo sirve de intermediario. (5) Todas esas operaciones en su conjunto hacen posible que un banco mantenga su estabilidad como entidad financiera.

1.2.3 Análisis de riesgo financiero

La principal función de los departamentos y/o áreas de riesgos crediticio es determinar el riesgo que significará para la institución otorgar un determinado crédito y para ello es necesario conocer a través de un análisis cuidadoso los estados financieros del cliente, análisis de los diversos puntos tanto cualitativos como cuantitativos que en conjunto permitirá tener una mejor visión sobre el cliente y la capacidad para poder pagar dicho crédito. (6)

Controlar la capacidad financiera de los clientes y analizar la factibilidad del otorgamiento de créditos, son algunas de las actividades desarrolladas por las entidades bancarias a la hora de llevar a cabo este tipo de análisis.

Riesgo de crédito

El riesgo de crédito se define como la volatilidad de los ingresos debido a pérdidas potenciales en crédito por falta de pago de un acreditado o contraparte, en otras palabras es el riesgo de que los clientes no cumplan con sus obligaciones de pago. (7)

Consiste en la probabilidad que ocurra un siniestro financiero por incapacidad de pago de los deudores del banco, tanto en forma individual como en forma consolidada. (8)

Es la posibilidad de que una entidad incurra en pérdidas y disminuya el valor de sus activos, como consecuencia de que sus deudores o contraparte fallen en el cumplimiento oportuno o cumplan imperfectamente los términos acordados en los contratos de crédito. (1)

El activo más importante y con mayor participación en una cooperativa que desarrolla actividad financiera es la cartera de créditos. Es el área principal de exposición de las cooperativas con sus asociados. El riesgo de crédito es la principal fuente de problemas en los entes financieros.

Entre los beneficios del análisis de riesgo de crédito se encuentran:

- Mejorar la rentabilidad de la institución (económica y social).
- Mejorar la calidad crediticia (minimizar cartera vencida).
- Eficiencia operativa.
- Generar reservas adecuadas y anticipar requerimientos de capital.

- Desarrollar plataforma para un sano crecimiento de la cartera crediticia (1)

1.3 Modelos para el análisis de riesgo de crédito

En la actualidad se contemplan dos tipos de modelos para la estimación del riesgo de crédito: los tradicionales y los de enfoque moderno. (9)

Modelo Tradicional

A partir de los años 60 transcurrieron varias décadas en las que el análisis de riesgo se realizaba a través de los llamados modelos tradicionales, que predicen la quiebra de las empresas a partir de las variables independientes (razones financieras, indicadores micro y macroeconómicos).

Los modelos tradicionales se basan en la experiencia del analista de riesgo que tomando la información financiera de la empresa y el historial de pago de la empresa solicitante, realiza el dictamen de riesgo.

El modelo tradicional más conocido es el de las cinco “C” del crédito (Carácter, Capital, Capacidad, Colateral y Ciclo), también llamado modelo experto, en el cual la decisión se deja en manos de un analista de crédito (experto), que analiza estos cinco factores claves. Implícitamente la experiencia de dicha persona, su juicio subjetivo y la evaluación de dichos factores, constituyen los elementos determinantes a la hora de otorgar o no el crédito.

Los elementos analizados por este modelo son los siguientes:

- **Carácter:** mide la reputación de la firma, su voluntad para pagar y su historial de pago, se ha establecido empíricamente que la antigüedad de creación de una empresa es un indicio adecuado de su reputación de pago.
- **Capital:** mide la contribución de los accionistas en el capital total de la empresa y la capacidad de endeudamiento, estos se ven como buenos indicios de la probabilidad de quiebra.
- **Capacidad:** mide la habilidad para pagar, la cual se refleja en la volatilidad de los ingresos del deudor, es decir en la viabilidad de las ganancias del acreditado. Se dice que el pago de su deuda sigue un patrón de constancia pero las ganancias son volátiles y puede haber períodos en los que disminuye la capacidad de pago de la empresa. Es muy significativo el

análisis de la capacidad de la gerencia, ya que de la misma depende en gran medida el logro de los objetivos de la empresa.

- **Colateral:** en casos de impagos, el banco tendría derecho sobre el colateral pignorado (dejado en garantía) por el deudor. Cuanto más prioritaria sea la reclamación, mayor es el valor de mercado del colateral correspondiente y menor la exposición al riesgo del crédito. Un colateral no convierte un mal crédito en bueno, pero si mejora uno bueno.
- **Ciclo económico:** elemento importante en la determinación de la exposición crediticia, sobre todo en aquellos sectores económicos que dependen de él. Es el único elemento a considerar que es ajeno al prestatario, comprende condiciones económicas y políticas generales.

El enfoque tradicional depende en gran medida de la información financiera presentada por la empresa solicitante del crédito. (9)

Modelos modernos

Los modelos de enfoque modernos se dedican a predecir cómo se verá la empresa en el futuro dado por los cambios en el mercado, considerando como elemento fundamental la competencia que existe entre empresas.

Dentro de los modelos de enfoque modernos de las últimas décadas se tienen:

- Modelos Z-Score:

Se estudian un conjunto de indicadores financieros que tienen como propósito clasificar a las empresas en dos grupos: bancarrota y no bancarrota. Se consideró que este método tiene una serie de limitaciones al utilizar razones financieras y estas tienen un efecto de subestimación en el tamaño de las estadísticas, es decir que en un análisis a través de razones financieras únicamente no se pueden identificar datos relevantes en el otorgamiento de un crédito.

- Modelo Zeta:

Constituye mejoras al modelo Z-Score ya que permite predecir la bancarrota de las empresas.

Existen otros modelos de enfoque moderno para el análisis de riesgo que tienen muy en cuenta el riesgo de mercado. (28)

- CreditRisk plus

Modelo de riesgo de crédito estadístico lanzado por Credit Suisse First Boston (CSFB) en 1997. (1) Se puede aplicar a cualquier tipo de producto de crédito, incluidos préstamos, bonos, cartas de crédito y derivados financieros. Para estimar el riesgo de crédito utiliza técnicas de análisis, en contraposición a las simulaciones. Cuando se estima el riesgo de crédito considera la calidad crediticia y el riesgo sistemático de los deudores. Se enfoca en la búsqueda de la probabilidad de incumplimiento. Permite sólo dos resultados, factible y no factible. También cuenta con aplicaciones para calcular disposiciones de riesgo de crédito, forzar el límite de crédito y gestionar operaciones de crédito.

- Credit Scoring

Es un modelo creado por la organización ACCION International (8). Para calcular el riesgo de otorgar el crédito utiliza la información histórica sobre las características del cliente y las variables relacionadas al crédito. Analiza la información sobre variables tales como las características demográficas, las características del microempresario y de la microempresa, y la historia de pago (si se tiene). Desarrolla una tarjeta de calificación (scorecard) que asigna puntos por cada factor que ayude a predecir el mérito crediticio del cliente. El número total de puntos - el *credit score* - ayuda a predecir la probabilidad de pago con base en la cuantificación de las características incluidas en la base de datos.

Análisis de Riesgo en Cuba

En Cuba los bancos que conceden créditos a empresas realizan el análisis de riesgo por medio del modelo tradicional. Introducen la información financiera de las empresas, dados por el balance general y el estado de resultados del año presente y del anterior, valorando en una página Excel el comportamiento de algunos de los indicadores financieros.

Cabe destacar que las principales desventajas que se plantean para este modelo son: la subjetividad que aparece producto al análisis de los expertos y la necesidad excesiva de analistas de riesgos en las entidades financieras. (9)

Se plantea que un analista de riesgos desprovisto de herramientas informatizadas puede ofrecer diferentes respuestas ante situaciones similares. El módulo a desarrollar constituye una herramienta para el analista de riesgos que ayuda sin lugar a dudas a disminuir la subjetividad, ya que por ser un sistema informatizado permite calcular la totalidad de los indicadores. Debido a lo anterior, se resalta

la importancia de proveerle al analista de riesgos herramientas informatizadas que le ayuden a tomar decisiones realizando análisis rápidos.

A pesar de que el modelo tradicional presenta una serie de deficiencias dadas por los problemas de consistencia y subjetividad, viéndose desplazado en muchos países por otras metodologías, en el BNC debido a la naturaleza del sistema socialista y subdesarrollado del país, se pretende continuar utilizando el enfoque tradicional, apoyando al analista de riesgo con herramientas informáticas.

1.4 Sistemas para el análisis de riesgo de crédito en el mundo

A nivel internacional se han venido desarrollando una serie de sistemas informáticos que permiten la gestión de los riesgos de las entidades. Entre ellos podemos encontrar los siguientes:

@Risk

- Ventajas:
  - Permite la gestión de gran número de riesgo, en cualquier tipo de entidad.
- Desventajas:
  - Usa el enfoque de los modelos modernos.
  - Basa el análisis de riesgo en la simulación, es software propietario, no es multiplataforma, no toma la información financiera de clientes para mostrar sus reportes y se basa en modelación de ambientes. (10)

Power Risk es un software de riesgos financieros creado por la empresa Scalar Consulting. (11) Entre los módulos que desarrolló está el Módulo “Power Risk–Scoring”.

- Ventajas:

Entre las funcionalidades que brinda este módulo están:

  - Generar puntajes para operaciones prospectivas por tipo de cartera.
  - Facilidad de generar variables cualitativas y numéricas por cada tipo de cartera.
  - Proveer probabilidades de incumplimiento ex-ante.
  - Incluye cálculo de margen de error.
  - Aplicable tanto a personas como empresas.
- Desventaja:
  - Es software propietario. Además, el pago de licencias es costoso.

Este módulo aporta una herramienta extremadamente útil y dinámica en la toma de decisiones, específicamente en la concesión de créditos nuevos, utilizando la experiencia misma de la institución. Además, permite racionalizar el proceso de crédito y aumentar la eficacia del asesor de crédito. (11)

El sistema @Risk presenta un inconveniente fundamental ya que al estar enfocado a los modelos modernos, no toma la información financiera de clientes para mostrar sus reportes por lo que, no se ajusta a los requerimientos del BNC para efectuar el análisis de riesgo de crédito.

De manera general los sistemas expuestos constituyen buenas soluciones informáticas para el mundo financiero, debido a la gama de funcionalidades que brindan, facilitando así la gestión de las operaciones bancarias. Por otro lado, las licencias de dichos productos constituyen una limitante; ya que son privativas y que poseen altos precios, lo que trae consigo que se dificulte tanto la adquisición como el uso y mantenimiento de los mismos.

1.5 Metodología definida para el desarrollo

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. (12)

Para el desarrollo de esta investigación se ha seleccionado la metodología RUP (en inglés: *Rational Unified Process*) (13) ya que la misma está enfocada a la producción de software con calidad.

1.5.1 Rational Unified Process

El Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. El Proceso Unificado es más que un simple proceso, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes niveles de aptitud y diferentes tamaños de proyecto. (14)

El Proceso Unificado utiliza el lenguaje de modelado UML (en inglés: *Unified Modeling Language*), para realizar todos los esquemas de un sistema de software, pero existen tres aspectos que realmente lo caracterizan y ellos son:

✓ **Dirigido por casos de uso:** Los casos de uso (CU, por sus siglas) describen lo que los usuarios futuros necesitan y desean, lo que es captado en el modelado del negocio y representado a través de los requerimientos. A partir de este momento todos los artefactos que se obtienen en el desarrollo representan la realización de los CU.

✓ **Centrado en la arquitectura:** La arquitectura provee al equipo de desarrollo y a los usuarios de una visión general del sistema en la que ambas partes deben de estar de acuerdo y describe los elementos del modelo que son más importantes para su construcción. La relación entre CU y arquitectura es estrecha, es decir los CU deben encajar en la arquitectura cuando se llevan a cabo mientras que la arquitectura debe permitir el desarrollo de todos los CU requeridos.

✓ **Iterativo e incremental:** El desarrollo de un proyecto supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo y los incrementos al crecimiento del producto. Para una efectividad máxima las iteraciones deben estar controladas.

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida del proyecto. Cada ciclo está dividido en cuatro fases: Inicio, Elaboración, Construcción y Transición y durante cada fase tienen lugar los siguientes flujos de trabajo: Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, Despliegue, Configuración y Mantenimiento del cambio, Administración del proyecto y Ambiente.

Debido a la naturaleza del presente trabajo se hace necesario centrarse en los flujos de trabajo de diseño e implementación.

✓ **Diseño:** En el flujo se describe cómo el sistema será realizado a partir de las exigencias previstas en el flujo de Requisitos, su función principal es tratar de interpretar y traducir los requisitos a una detallada descripción que indique a los implementadores cómo desarrollar el sistema, obteniendo como principal resultado el modelo de diseño.

✓ **Implementación:** En la implementación se comienza con los resultados del diseño e implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. El propósito general de este flujo es desarrollar la arquitectura y el sistema como un todo, obteniendo como resultado principal el modelo de implementación. (14)

1.5.2 Lenguaje de modelado

Se denomina lenguaje de modelado de objetos al conjunto estandarizado de símbolos y las distintas combinaciones de ellos para modelar un diseño de software. (15)

UML: Es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, soportando además el paradigma orientado a objetos. Se caracteriza por dividir a los sistemas en una estructura estática y un comportamiento dinámico. (16) Básicamente facilita a los desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados.

1.6 Patrones de diseño

Hoy en día se ha hecho muy común el uso de los patrones cuando se quiere desarrollar un software. Su empleo brinda a los equipos de desarrollo un elemento que agiliza el diseño del sistema, debido a que establecen soluciones a problemas comunes del diseño, facilitan la reutilización del código y permiten una fácil comprensión debido a la documentación estándar que brindan.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores. (16) Existen dos grupos principales en los que estos se pueden agrupar.

1.6.1 Patrones GRASP

Los Patrones Generales para Asignar Responsabilidades (GRASP, por sus siglas en inglés), describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, su nombre se debe a la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

GRASP destaca 5 patrones principales: Experto, Creador, Bajo acoplamiento, Alta cohesión, Controlador.

Patrón Experto: consiste en asignar una responsabilidad al experto en información, es decir a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Refuerza el encapsulamiento y favorece el bajo acoplamiento.

Patrón Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Su propósito principal es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.

Patrón Bajo Acoplamiento: pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir forma un diseño de clases más independientes, sin que se relacionen con muchas otras, reduciendo así el impacto de los cambios, haciéndolos más reutilizables y acrecentando la oportunidad de una mayor productividad.

Patrón Alta Cohesión: su objetivo es asignar una responsabilidad de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Patrón Controlador: consiste en asignar la responsabilidad a una clase de manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. (16)

1.6.2 Patrones GoF

Entre los patrones propuestos por la pandilla de los cuatro (en inglés: Gang Of Four) se pueden encontrar los que se explican seguidamente.

Patrón Fachada: patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace las veces de pantalla o fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezca un punto de entrada al sistema tapado por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.

Patrón Mediador: patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.

Patrón Cadena de Responsabilidad: se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición.

Patrón Singleton: patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones. (16)

1.7 Herramientas y tecnologías utilizadas

La utilización de los frameworks, lenguajes y herramientas estudiados como parte del ambiente de desarrollo, fueron definidas por el equipo de arquitectos del proyecto SAGEB según lo indicado en (13), con el fin de lograr un desarrollo ágil con el menor coste posible y la obtención de un producto como resultado final, con la calidad requerida.

1.7.1 Lenguajes de programación en el lado del servidor

Java: es un lenguaje de programación orientado a objetos, que permite a los programadores realizar aplicaciones de múltiples tipos, ya sean de escritorio o web. Se caracteriza por ser un lenguaje simple, robusto y poderoso que se torna fácil de aprender, debido a que elimina sentencias de bajo nivel además del Recolector de Basura, haciendo transparente para los programadores el manejo de la memoria. Se destaca por ser un lenguaje de código abierto y multiplataforma por lo cual ha logrado una gran expansión por todo el mundo. En la actualidad incluye un gran número de librerías para múltiples trabajos. (17)

Plataforma JEE (Java Enterprise Edition en inglés)**:** define estándares para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java, empleando arquitecturas que definen un modelo multicapa y que se apoyan en componentes de *software* modulares. JEE incluye tecnologías, tales como Servlets, JSP (en inglés: Java Server Pages) y varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras a la vez que son integrables con tecnologías anteriores. (18)

1.7.2 Lenguajes de programación en el lado del cliente

XHTML (en inglés: Extensible Hypertext Markup Languaje) es el lenguaje de marcado pensado para sustituir a HTML (en inglés: Hypertext Markup Languaje), es una reformulación del mismo que es compatible con XML (en inglés: Extensible Markup Languaje) Se utiliza para generar documentos y

contenidos de hipertexto generalmente publicados en la web. El documento se escribe en forma de etiquetas, que hasta cierto punto pueden establecer la apariencia del documento, aunque en la actualidad se suele mezclar con lenguajes script como JavaScript, para lograr mayor interacción con los usuarios. (19)

JavaScript: Es un lenguaje interpretado, es decir que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el XHTML. JavaScript no es orientado a objetos aunque permite la creación de objetos propios. Se caracteriza por ser un lenguaje manejado por eventos por el hecho de responder a eventos generados ya sea por el usuario o por el navegador, es independiente de la plataforma debido a que solo se necesita un navegador para ejecutar el código, permite un desarrollo rápido y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox. (20)

1.7.3 Frameworks

Un framework (marco de trabajo en español) es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Promueve la reutilización de código, con el fin de ahorrarle trabajo al desarrollador al no tener que reescribir ese código para una nueva aplicación que desee crear. En un sentido muy amplio el framework que se utiliza determina la arquitectura del software. (21)

Spring: es un framework bajo licencia de código abierto concebido para el desarrollo de aplicaciones basadas en la plataforma Java/JEE. Spring ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto, haciendo uso de las prácticas comunes en la industria de software. Su funcionamiento se basa en inversión de control e inyección de dependencia. Se hace uso de Spring Framework en su versión 2.5. El framework está dividido en módulos para su correcto funcionamiento, ellos son:

✓ **Spring Core:** representa el núcleo de Spring, es donde se encuentran funcionalidades fundamentales que provee el framework.

✓ **Spring AOP:** ofrece un extenso soporte para Programación Orientada a Aspectos, permitiendo definir entre otras cosas la política transaccional y de seguridad de una aplicación.

✓ **Spring ORM:** brinda el soporte necesario para la integración con el framework Hibernate.

✓ **Spring DAO[1]:** provee un trabajo con JDBC (Java Database Connectivity en inglés) en aras de hacer el código de acceso a datos más limpio y entendible, ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos.

✓ **Spring Web:** es el encargado de crear el contexto para aplicaciones web, incluye soporte para una variedad de tareas como la subida de archivos, la vinculación de parámetros de las peticiones a objetos del negocio, etc.

✓ **Spring Web MVC** (Modelo-Vista-Controlador en español): ofrece gran soporte para el desarrollo de aplicaciones web utilizando la filosofía MVC, emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio.

✓ **Spring Context:** es el que consagra a Spring como un framework, ofrece soporte para la internacionalización, la aplicación de eventos de ciclo de vida. Brinda soporte a servicios como correo electrónico, acceso a Java Naming and Directory Interface (JNDI, por sus siglas en inglés), integración con Enterprise JavaBeans (EJB, por sus siglas en inglés). (22)

Spring Web Flow: es un framework y a su vez un subproyecto de Spring que permite controlar la navegación de la aplicación web, guiando al usuario a través de una serie de pasos para completar una transacción de aplicación. Los flujos web son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas. En Spring Web Flow un flujo se define mediante un archivo XML en el que se definen las reglas. Permite la creación de flujos reutilizables en toda la aplicación. (22) Se hace uso de Spring Web Flow en su versión 1.9.7.

Hibernate Framework: es una solución ORM (en inglés: Object Relacional Mapping) para el lenguaje de programación Java, concebido bajo la filosofía de código abierto. Busca solucionar el problema de la diferencia entre el modelo de objetos y el modelo relacional, realizando un mapeo entre tablas de la base de datos y los objetos del negocio a través de archivos declarativos, en este caso archivos XML. Soporta la conexión a una gran variedad de servidores de base de datos, como PostgreSQL, Oracle, SQL Server y otros. Permite además adaptarse a una base de datos ya existente, así como generar la base de datos a partir de un modelo objetual. Admite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos propiamente

[1] Para más detalle ver Capítulo 2 Epígrafe 2.2.1

dicho, aunque permite la ejecución de consultas SQL (en inglés: Standrad Query Languaje). (23)Se hace uso de Hibernate Framework en su versión 3.5.

Dojo Toolkit: es una colección de scripts estáticos que permiten el desarrollo de aplicaciones web enriquecidas en el cliente. Incorpora soporte para el trabajo con la tecnología AJAX (en inglés: Asynchronous JavaScript and XML). Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten. Hace transparente el desarrollo para diferentes implementaciones del DOM (en inglés: Document Object Model), ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS (Cascading Style Sheet, en inglés) y JavaScript para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: Dojo, Dijit y Dojox. (24) Se hace uso de Dojo Toolkit en su versión 1.3.

1.7.4 Herramientas de desarrollo

Herramientas CASE (en inglés: Computer Aided Software Engineering): consisten en diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas. Este tipo de herramientas ayuda en todos los aspectos del ciclo de vida del desarrollo del software como la realización de su diseño, generación de código y de documentación a partir de un diseño dado. En sentido general estas herramientas intentan dar ayuda automatizada al proceso de desarrollo de software y sirven de apoyo a la metodología de desarrollo empleada. (25)

Visual Paradigm: herramienta multiplataforma para el modelado UML que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Su principal dificultad radica en que posee una licencia muy restringida. (26)Se hace uso del Visual Paradigm en su versión 6.4.

Control de versiones: se le llama así a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Un sistema de control de versiones debe proporcionar un

mecanismo de almacenaje de los elementos que deba gestionar y un registro histórico de las acciones realizadas con cada elemento o conjunto de elementos. (27)

Subversion: es un sistema de control de versiones libre y de código abierto conocido habitualmente como SVN que se ha expandido en gran medida dentro de la comunidad del desarrollo de software. Maneja ficheros y directorios a través del tiempo y la información radica en un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas en ordenadores distintos, fomentando la colaboración que deriva en la reducción del tiempo de desarrollo. (27) Se hace uso de subversion en su versión 1.6.6.

Entorno integrado de desarrollo

Eclipse: es una herramienta de código abierto y multiplataforma desarrollado por la compañía IBM (International Business Machines en inglés). Emplea plug-ins o complementos para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura de plug-ins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías. (28) Se utiliza el Eclipse IDE en su versión 3.4.

Servidor de aplicaciones web

Tomcat: es un contenedor de servlets bajo la filosofía del código abierto licenciado con Apache Sofware License que presenta la ventaja de ser multiplataforma. Implementa las especificaciones de Servlets 2.5 y JSP 2.1. Con frecuencia se presenta en combinación con el servidor web Apache aunque puede realizar esta función por sí mismo. En la actualidad es utilizado como un servidor web autónomo en entornos donde existe un alto nivel de tráfico y alta disponibilidad. (29) Se hace uso de Tomcat en su versión 6.0.

Servidor de Base de Datos

Microsoft SQL Server 2005: es un servidor de base de datos basados en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración. Permite además trabajar en modo cliente-servidor.

Promueve la escalabilidad y seguridad de la información. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés: Transact-SQL). (30) Requiere para su funcionamiento un sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización.

1.8 Conclusiones Parciales

Una vez finalizado el presente capítulo se pudo arribar a las siguientes conclusiones:

- Luego del análisis realizado se detectó que los sistemas de gestión bancaria estudiados no cumplen con los requerimientos necesarios para realizar el proceso de análisis de riesgo en el BNC.
- El ambiente de desarrollo integrado por la metodología seleccionada, además de los lenguajes, tecnologías y herramientas definidas, posibilitarán el desarrollo de una solución que apoye el cumplimiento de los principios de soberanía e independencia tecnológica.

CAPÍTULO 2: DISEÑO Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

2.1 Introducción

La principal aspiración de este capítulo es transformar los requisitos funcionales en el diseño de la solución propuesta, que permita un mejor entendimiento de los mismos, en aras de conformar una entrada adecuada para la posterior implementación. En consecuencia, se obtendrá un conjunto de artefactos que serán de gran valor para la fase de construcción como son: el diagrama de paquetes, los diagramas de clases, los diagramas de secuencia, y el modelo de datos. Con el fin de desarrollar un diseño robusto y sencillo se realiza la validación del mismo a partir de la utilización de métricas.

2.2 Arquitectura definida para el proyecto

Uno de los elementos clave en todo proceso de desarrollo de software es la definición de la arquitectura. Esta representación eleva el nivel de abstracción, facilitando así la comprensión de sistemas de software complejos, asimismo hace que aumenten las posibilidades de reutilizar tanto la arquitectura como los componentes que aparecen en ella. En la medida que sea concebida la arquitectura basada en los principios de cohesión, utilidad y flexibilidad de los componentes se obtendrá un mejor acabado del producto.

“La arquitectura del software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución”. (31)

2.2.1 Características de la arquitectura del sistema

A continuación se explican los principales componentes que conforman la arquitectura del sistema informático Quarxo, que tiene como primer elemento el desarrollo bajo la tecnología JEE. La arquitectura base está compuesta por tres capas: Presentación, Negocio y Acceso a Datos y una capa transversal a las otras con las clases del dominio. (13) Para más detalles ver Anexo 1: Representación de la arquitectura del sistema.

✓ **Capa de Presentación:** en esta capa se desarrollan las interfaces pertenecientes a la presentación. En el lado del cliente se utiliza la librería Dojo Toolkit para generar las interfaces que interactúan con el usuario. En el lado del servidor se emplea Spring MVC para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente y también se utilizará Spring Web Flow

para representar y controlar los flujos complejos y reutilizables de la aplicación. La capa de presentación está relacionada con la capa de negocio y la capa de dominio.

✓ **Capa de Negocio:** está divida en dos subcapas: facade (fachada, en español) y manager (manejador o administrador, en español). La facade es el punto de intercambio entre la capa de presentación y la capa de negocio. Esta capa no tiene lógica de negocio, sino que agrupa las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación. La subcapa facade delega a la subcapa manager la realización de la lógica del negocio.

Por otro lado, la subcapa manager tiene la jerarquía de clases suficiente para implementar el negocio de la aplicación. Esta subcapa utiliza la capa de acceso a datos para obtener los datos persistidos y la capa de dominio para generar las entidades del negocio.

Desde la capa de negocio se envuelven transaccionalmente las funcionalidades de la aplicación para evitar inconsistencia en los datos. Se utiliza para esto, las políticas de transacciones que propone Spring Framework. Se utiliza el contenedor de Spring Framework para declarar y representar las relaciones de dependencia de cada una de las clases, el módulo Spring Security para asegurar las invocaciones a los métodos, el módulo Spring AOP para ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio.

✓ **Capa de Acceso a Datos:** En esta capa se encuentran las operaciones que permiten la interacción con el gestor de base de datos desde la aplicación. Desde aquí se ejerce la conexión con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información. La interacción con la capa de negocio se realiza a través de interfaces. Se utiliza el patrón DAO (Data Access Object, por sus siglas en inglés) para el desarrollo de la capa, el framework de persistencia Hibernate y los módulos de Spring: Spring ORM y Spring DAO. Hibernate como tecnología ORM para el manejo objetual de acciones de persistencia con la base de datos. Spring ORM y Spring DAO para soportar la integración con Hibernate y la utilización del patrón DAO.

La arquitectura definida presenta como otra de sus características la utilización del patrón MVC; patrón que propone dividir la aplicación en tres capas distintas, el Modelo, la Vista y el Controlador, potenciando la flexibilidad y la adaptabilidad a futuros cambios. Específicamente el modelo es la representación de la información que maneja la aplicación, la vista constituye la representación del modelo en forma gráfica disponible para la interacción con el usuario y el controlador se encarga de

responder a las solicitudes del usuario desde la interfaz, manejando los diferentes eventos a través de las funcionalidades necesarias y la información perteneciente al modelo.

2.2.2 Estructura del sistema

El sistema Quarxo está estructurado a través de subsistemas, módulos y componentes, definidos según las funcionalidades identificadas en la captura de requisitos. Los subsistemas agrupan un conjunto de módulos relacionados con los procesos que ejecutan. Los módulos agrupan un conjunto de CU que representan uno o más procesos bancarios estrechamente relacionados. Los componentes son un conjunto de funcionalidades comunes que son reutilizados por otros módulos del sistema. (13)

2.3 Análisis de los artefactos de entrada al diseño

Anteriormente se elaboró un trabajo de diploma en el cual se realizó el análisis y diseño del módulo Análisis de riesgo de crédito. (3) El cliente trazó nuevos requerimientos, mostrados más adelante dentro de este mismo epígrafe, por lo cual a pesar de que existen similitudes entre el diseño anterior y el que se propone, se decidió rediseñarlo, tomando como referencia el anterior.

Es importante destacar que aunque RUP contempla análisis y diseño en la misma disciplina por estar muy relacionadas son actividades diferentes, con artefactos diferentes según lo indicado por (14).

Un elemento clave para el diseño y la posterior implementación de la solución es la especificación de requisitos, donde se tienen en cuenta las funcionalidades que el sistema debe brindar. En este caso se tomarán como entrada los requisitos funcionales.

Los requisitos funcionales representan el comportamiento que tendrá el sistema. Describen las funcionalidades que debe cumplir o que se espera que este provea. Deben ser lo más completo, claro y conciso posible. Además pueden definirse a partir de reglas del negocio o la propia interacción de los usuarios, según lo indicado por (32)

A continuación se presentan los requisitos funcionales que han sido definidos previamente para el módulo Análisis de riesgo de crédito.

RF 1: Registrar información financiera.
RF 2: Actualizar información financiera.
RF 3: Consultar información financiera.

RF 4: Eliminar información financiera.
RF 5: Buscar información financiera.
RF 6: Registrar indicadores contables.
RF 7: Priorizar indicadores contables.
RF 8: Actualizar indicadores contables.
RF 9: Consultar indicadores contables.
RF 10: Eliminar indicadores contables.
RF 11: Buscar indicadores contables.
RF 12: Registrar dictamen de riesgos.
RF 13: Actualizar dictamen de riesgos.
RF 14: Consultar dictamen de riesgos.
RF 15: Eliminar dictamen de riesgos.
RF 16: Buscar dictamen de riesgos.

Los diagramas de casos de uso (DCU) del sistema constituyen una representación de los requerimientos funcionales, los roles que interactúan con el sistema y las interfaces hacia sistemas externos. En el módulo Análisis de riesgo de crédito se definió un actor y un total de dieciséis requisitos funcionales, los cuales han sido agrupados en tres CU como se muestra en la imagen siguiente.

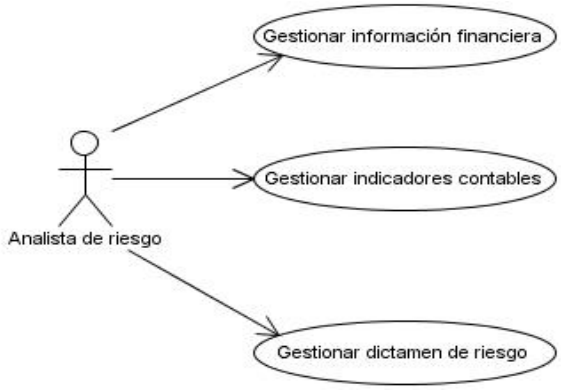


Figura 1: Diagrama de CU del módulo Análisis de riesgo de crédito.

Un actor del sistema es la persona que inicializa un CU del sistema, es decir, que interactúa con el mismo, es por esto que se define como actor del sistema al Analista de riesgo. Este es el encargado de registrar, organizar y almacenar toda la información que se obtiene y se genera en todo el proceso de otorgamiento del crédito.

Para entender la funcionalidad asociada a cada CU no es suficiente con la representación gráfica del DCU la misma debe ir acompañada de una descripción textual, que puede ser elaborada de forma breve o extendida y debe ir acompañada del prototipo respectivo. El prototipo del sistema que se construye en este punto, da una visión de las interfaces diseñadas para cada CU. Se presentan las descripciones detalladas de los CU del módulo Análisis de riesgo de crédito en el Anexo 2: Descripción de los CU del módulo Análisis de riesgo de crédito.

Requisitos no funcionales.

Los requisitos no funcionales establecen las características o propiedades que debe tener el sistema. Muchos requerimientos no funcionales se refieren al sistema como un todo más que a rasgos particulares del mismo. El incumplimiento de un requisito funcional puede degradar un sistema, sin embargo una falla de un requisito no funcional puede inutilizar un sistema, según lo indicado por (32).

Funcionalidad:

RNF 1 El sistema mostrará los errores en forma de mensajes.

- Todos los mensajes de error del sistema deberán incluir una descripción textual del error.

Usabilidad:

RNF 2 Los formularios serán estandarizados, por tanto:

- Los campos de texto tendrán un tamaño estándar de acuerdo con el espacio que se tenga en el área de la página y en la medida que se llene esa área primaria agregar la barra de desplazamiento vertical.
- No se utilizarán textos extensos para las etiquetas de la interfaz de usuario.

RNF 3 El menú de navegación estará disponible en todas las páginas.

RNF 4 En caso de que los resultados de las consultas tengan más de 10 coincidencias, estos se mostrarán de forma paginada en una tabla.

- Se mostrará en la parte inferior de la tabla el total de elementos encontrados, enlaces de navegación: ir hacia delante, hacia atrás o ir al inicio de los resultados mostrados.

Fiabilidad:

RNF 5 El sistema estará disponible durante toda la jornada laboral del BNC.

Seguridad:

RNF 6 El sistema permitirá la visualización de la información según el usuario autenticado.

RNF 7 El sistema implementará el uso de campos obligatorios y validaciones para garantizar la integridad de la información que se introduce por el usuario.

RNF 8 No se permitirá al usuario acceder a recursos de la aplicación sin identificarse.

Restricciones de diseño:
RNF 9 El sistema se implementará usando la plataforma JEE.
RNF 10 El sistema estará basado en un estilo arquitectónico en capas.

Interfaz de Usuario:
RNF 11 Todos los textos y mensajes en pantalla aparecerán en idioma español. Los errores serán visibles al usuario y en lo posible incluirán sugerencias de las posibles soluciones.
RNF 12 El sistema presentará los términos capitalizados, es decir, tendrán su primera letra en mayúsculas.

2.4 Diseño de la solución

De forma general en el diseño se modela el sistema, contribuyendo a una arquitectura estable y sólida, para que soporte todos los requisitos, tanto funcionales como no funcionales. El diseño posibilita una entrada apropiada y un punto de partida para las actividades de la implementación, descompone los trabajos de implementación en partes más manejables que pueden ser llevados a cabos por diferentes equipos de desarrollo. (3)

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales junto a otro grupo de restricciones relacionadas con el entorno de implementación tienen impacto en el sistema, está compuesto por otros artefactos como los diagramas de paquetes y diagramas de clases, además sirve como abstracción a la implementación y se empleará como entrada fundamental de las actividades de esta etapa. (14)

Los paquetes permiten describir la arquitectura de un sistema haciendo uso de la notación UML. Pueden mostrar el particionamiento del sistema, indicando cuáles clases o casos de uso se agrupan en un subsistema, y se conocen como paquetes lógicos. Los paquetes también pueden tener relaciones, de manera similar a los diagramas de clases, que podrían incluir asociaciones y herencia, según (33).

Los diagramas de paquetes expresan cómo está dividido el sistema en agrupaciones lógicas, mostrando las dependencias entre estas, según (16). Durante el desarrollo de software resulta muy conveniente agrupar clases y ficheros por diferentes criterios que ayudarán a una fácil comprensión de la aplicación.

A continuación se muestra la estructura y dependencia de paquetes del módulo Análisis de riesgo de crédito y una explicación de la composición de cada uno de ellos.

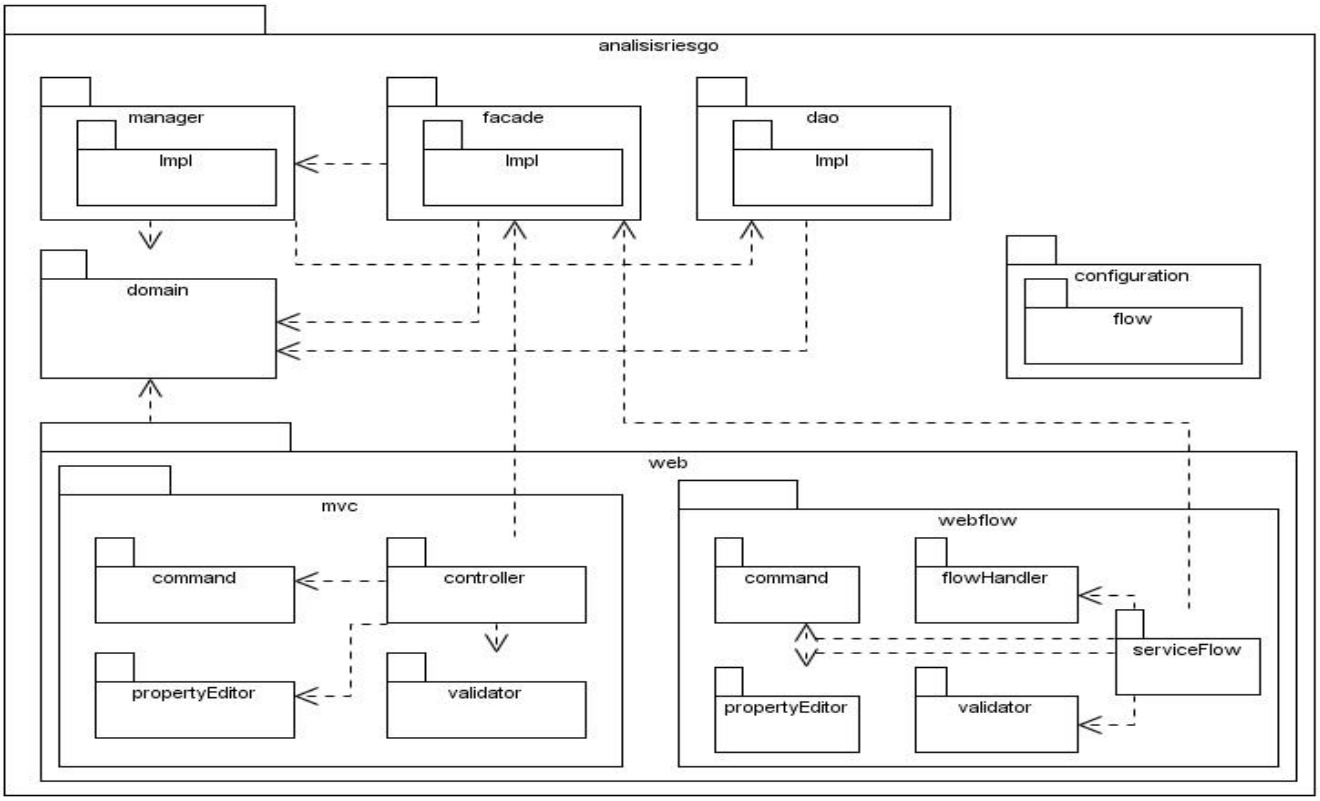


Figura 2: Diagrama de paquetes del módulo Análisis de riesgo de crédito.

***configuration*:** en este paquete se encuentran los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo, ellos son:

- servlet.xml: Define el contexto de Spring MVC.
- bussiness.xml: Define el contexto para el negocio.
- webflow.xml: Define el contexto para Spring Web Flow.
- dataaccess.xml: Define el contexto para acceso a datos.

***flow*:** están presentes los archivos XML que definen los flujos de la aplicación.

***manager*:** se hallan las interfaces y las implementaciones que comprenden la lógica de negocio que existe en el módulo, que será brindada a las capas superiores.

***facade*:** se encuentran la interfaz y su respectiva implementación, encargadas de brindar las funcionalidades que serán usadas por la presentación, así como las que se consumen por otros módulos.

***dao*:** se encuentran igualmente las interfaces y las implementaciones encargadas de brindar los servicios de comunicación con la base de datos, además de hallarse también los ficheros de mapeo utilizados por Hibernate correspondientes al módulo.
***domain*:** aquí se localizan las clases relacionadas con el dominio del módulo en cuestión.
***web*:** agrupa un conjunto de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.
***mvc*:** en este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring MVC.
***webflow*:** en este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring Web Flow.
***controller*:** se hallan las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.
***serviceFlow*:** contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.
***flowHandler*:** dispone de clases utilizadas para personalizar el trabajo con el Web Flow.
***command*:** se localizan las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.
***propertyEditor*:** agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.
***validator*:** cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor.

Una realización de CU del diseño es una colaboración del modelo de diseño que describe cómo se realiza un CU específico y cómo se ejecuta en término de clases de diseño y sus objetos. Una realización contiene diagramas de clases que muestran sus clases de diseño participantes y diagramas de interacción que muestran la realización de un flujo o escenario en concreto, en términos de interacción entre objetos. (14). Los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación.

A continuación se muestra el diagrama de clase realizado para el escenario Registrar información financiera del CU Gestionar información financiera, el cual se tomará como objeto de estudio por ser un requisito primordial del módulo. El resto de los diagramas de clases se encuentran en el Anexo 3: Diagramas de clases del módulo Análisis de riesgo de crédito.

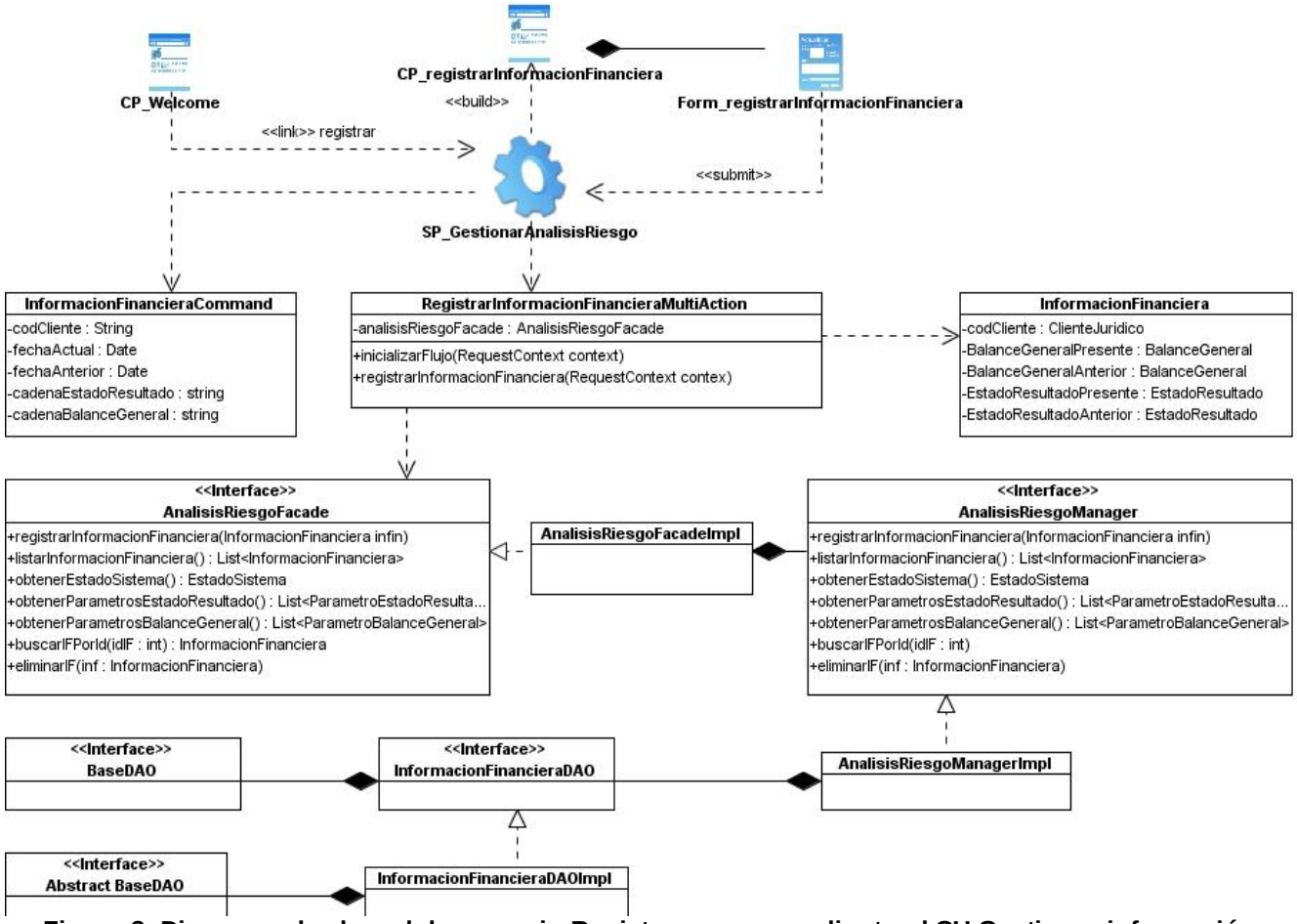


Figura 3: Diagrama de clase del escenario Registrar correspondiente al CU Gestionar información financiera.

Los diagramas de clases expuestos en la capa de presentación, se hicieron utilizando estereotipos web. Para un mejor entendimiento de los mismos, a continuación se brinda una breve descripción de dichos estereotipos y seguidamente, de las clases empleadas.

Client Page (página cliente): páginas web encargadas de mostrar los formularios e información al usuario.

Form (formulario): colección de elementos de entrada que son parte de una página cliente.

Server Page (página servidora): representa la página web que tiene código, que se ejecuta en el servidor.

<<***Build***>>*:* representa una asociación especial que relaciona las páginas cliente con las páginas servidor.

<<***Link***>>*:* expresa las asociaciones más comunes entre las páginas, en este caso la del hipervínculo.
<<***Submit***>>*:* es la relación que se crea siempre entre una página servidor y un formulario.

GestionarAnalisisRiesgo: clase que representa el mecanismo interno con el que Spring Web Flow gestiona las peticiones realizadas desde el cliente.
RegistrarInformacionFinancieraMultiAction: clase que gestiona las acciones asociadas al flujo de registrar alguna información financiera. Spring Web Flow interactúa con ella para su funcionamiento.
InformacionFinanciera: clase que representa la información que persiste en la base de datos asociada a la información financiera.
InformacionFinancieraCommand: representa el modelo del que se estará haciendo uso en la vista.
InformacionFinancieraDAO: interfaz que brinda las funcionalidades correspondientes al manejo del acceso a datos para la información financiera.
InformacionFinancieraDAOImpl: clase que implementa la interfaz InformacionFinancieraDAO. En ella se implementa el manejo del acceso a datos correspondiente a alguna información financiera.
AnalisisRiesgoFacade: interfaz de la capa de lógica de negocio que contiene las funcionalidades relacionadas con el sistema, brindadas a la capa de presentación y a la vez le sirve de fachada.
AnalisisRiesgoFacadeImpl: clase que implementa las funcionalidades definidas en la interfaz AnalisisRiesgoFacade. En esta no se implementa ninguna lógica de negocio, simplemente se limita a delegar las responsabilidades a la clase Manager correspondiente.
AnalisisRiesgoManager: interfaz que define las funcionalidades de lógica de negocio específicas para el módulo Análisis de riesgo de crédito.
AnalisisRiesgoManagerImpl: clase que implementa los métodos de lógica de negocio definidos en la interfaz AnalisisRiesgoManager. Se encarga de implementar las funcionalidades brindadas por la fachada.
BaseDAO: interfaz que agrupa funcionalidades comunes relacionadas con los métodos de acceso a datos.
AbstractBaseDAO: esta clase abstracta, que no implementa ninguna lógica de acceso a datos, sólo se encarga de delegar esas responsabilidades a los DAOs.

2.4.1 Patrones de diseño empleados

El diseño fue elaborado empleando patrones basados en la experiencia de los desarrolladores. De

manera general, constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. En este caso se emplearon los patrones GRASP y GoF que se exponen posteriormente.

Entre los patrones GRASP empleados están:

✓ **Controlador:** la clase controladora RegistrarInformacionFinancieraMultiAction, constituye un ejemplo de la aplicación de este patrón, la misma tendrá en cuenta la responsabilidad de manejar los eventos que consisten en preparar los datos que serán mostrados al usuario.

✓ **Experto:** se aplica en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan. Específicamente: la clase InformacionFinancieraDAO, será la responsable de efectuar las operaciones que conciernen a las funciones: insertar, eliminar, actualizar y consultar la información financiera. Sobre este mismo principio se realiza el diseño de las restantes funcionalidades.

✓ **Alta cohesión:** este patrón fue utilizado en el diseño de manera general; donde se agruparon las clases en dependencia de los requerimientos del módulo a los que se les debía dar respuesta, según la premisa de que cada clase debe implementar las operaciones que estén sobre la misma área funcional.

✓ **Bajo acoplamiento:** se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz AnalisisRiesgoFacade y su implementación, que permiten que RegistrarInformacionFinancieraMultiAction, clase de la presentación, se relacione únicamente con ella para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

Se emplearon además los siguientes patrones GoF:

✓ **Fachada:** la utilización de este patrón se manifiesta en la definición de la interfaz AnalisisRiesgoFacade y su implementación, responsables de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

✓ **Cadena de Responsabilidad:** cuando desde la vista se solicita información que se encuentra en la base de datos, esta es atendida primeramente por los controladores, luego por la fachada, el manejador y finalmente el DAO, evidenciándose de esta manera la utilización de dicho patrón.

2.4.2 Diagramas de interacción

Los diagramas de interacción son importantes para modelar los aspectos dinámicos de un sistema y para construir sistemas ejecutables por medio de ingeniería directa e inversa. Muestran la realización de un flujo o escenario concreto de un CU en términos de interacción entre objetos del diseño. Los diagramas de secuencia son un ejemplo de diagramas de interacción, que destacan principalmente el orden temporal de los mensajes para varios escenarios del módulo en cuestión. Muestran las interacciones entre objetos mediante transferencias entre objetos y subsistemas, según lo indicado por (14). A continuación se presenta el diagrama de secuencia asociado al escenario Registrar correspondiente al CU Gestionar información financiera. Ver otros en el Anexo 4: Diagramas de secuencia del módulo Análisis de riesgo de crédito.

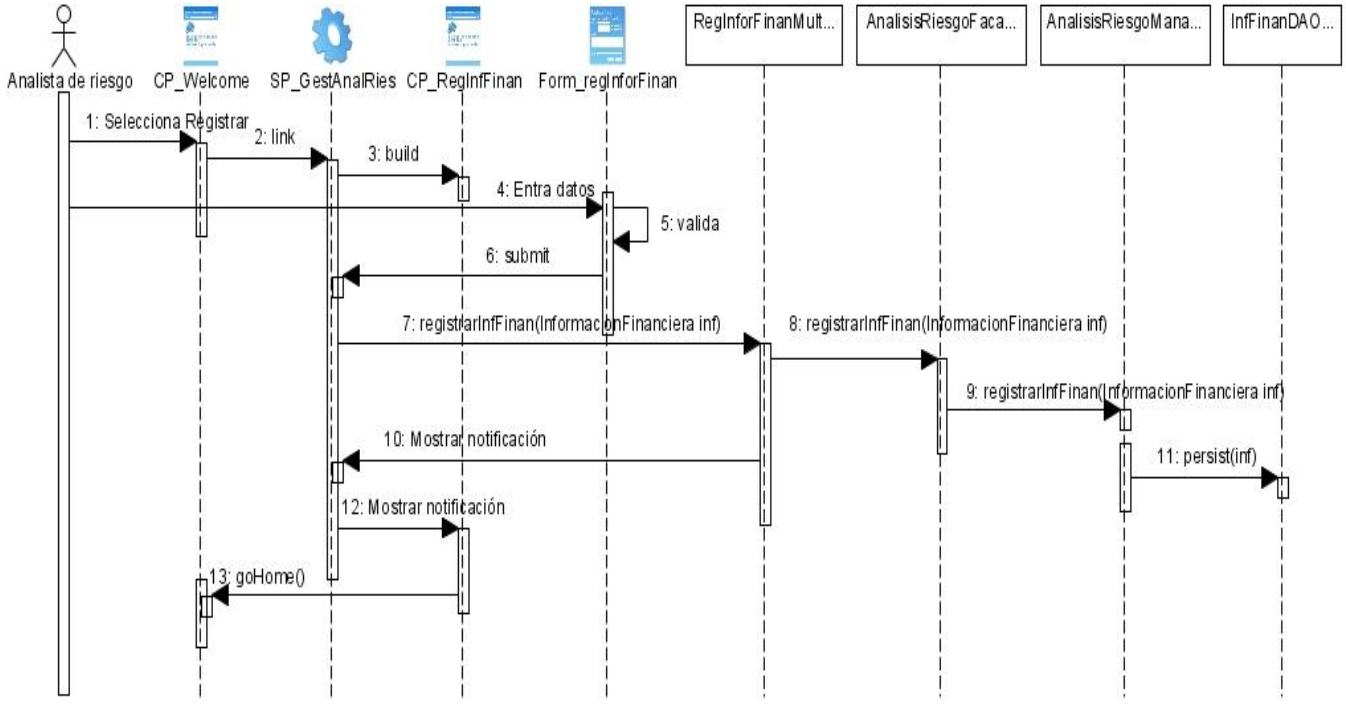


Figura 4: Diagrama de secuencia del escenario Registrar correspondiente al CU Gestionar información financiera.

2.4.3 Modelo de datos

Teniendo en cuenta los requisitos funcionales del módulo Análisis de riesgo de crédito, se hizo necesario crear un modelo de datos para la representación de los datos persistentes que son manejados por el sistema.

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, permite describir las estructuras de datos de la base (el tipo de los datos que incluye la base y la forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base). (34)

En la figura 5 se puede observar el modelo de datos que se realizó para representar los datos que son manipulados en el proceso de análisis de riesgo de crédito.

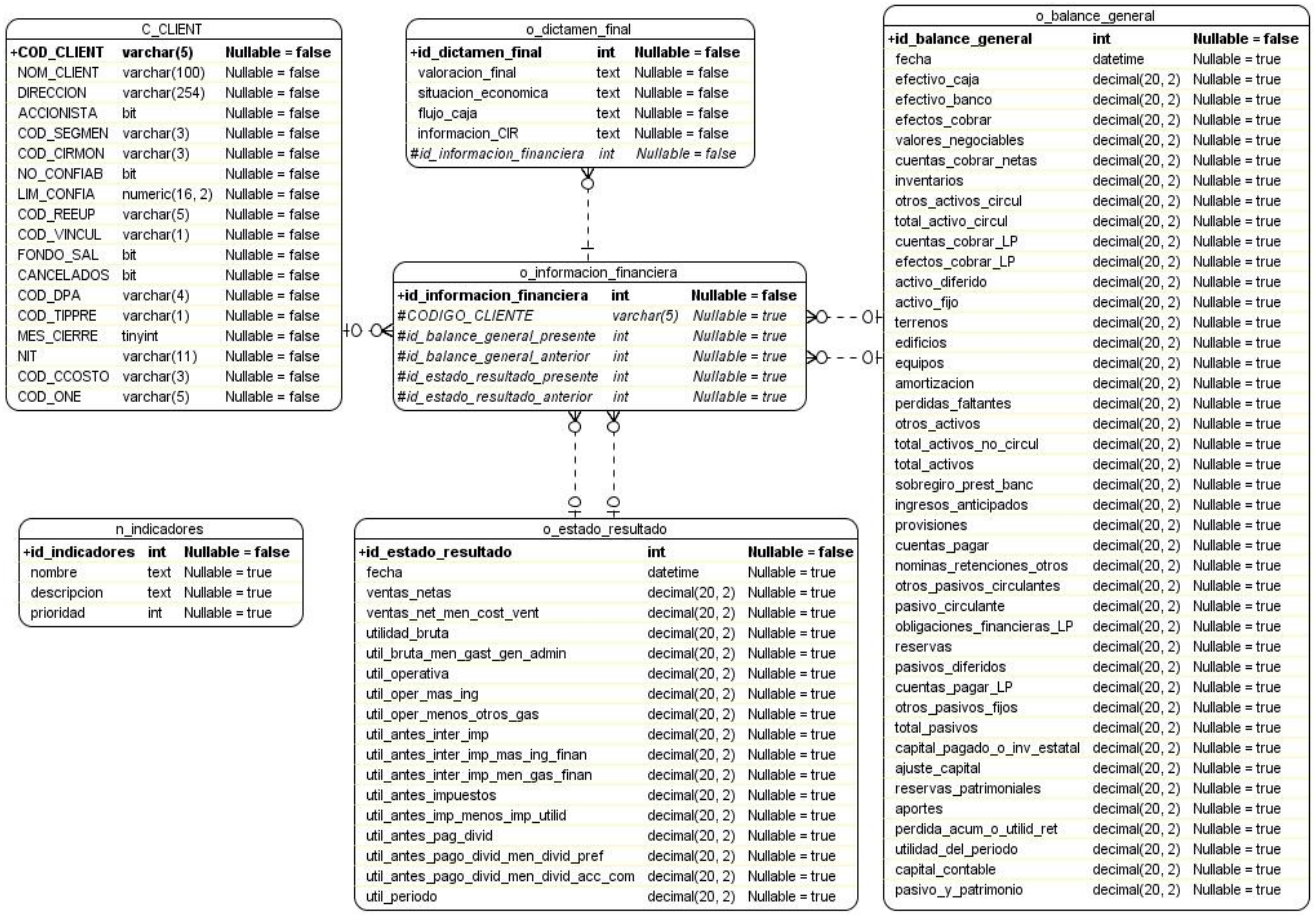


Figura 5: Modelo de datos correspondiente al módulo Análisis de riesgo de crédito.

Se creó la tabla principal o_informacion_financiera en la que se almacenarán los datos de la información financiera una vez que es registrada en el sistema. Esta a su vez posee una relación con las tablas o_balance_general y o_estado_resultado, que guardarán los balances y los estados de resultados anteriores y actuales respectivamente. Es necesario destacar que estas tablas son muy importantes para el proceso que se quiere realizar ya que el análisis de riesgo de crédito depende en

gran medida de la información financiera presentada por el cliente, la cual es analizada en base al balance general y el estado de resultado del año presente y del anterior, lo que justifica la relación que existe entre estas tablas.

La tabla C_Client, ya creada en el sistema Quarxo, tiene una relación con la tabla o_informacion_financiera, que permitirá asociar una información financiera a un cliente determinado.

Para guardar el dictamen de riesgo creado a partir de la información financiera presentada por un cliente, se decidió introducir la tabla o_dictamen_final, que almacenará el documento elaborado como resultado del análisis de riesgo de crédito realizado.

El nomenclador n_indicadores se realizó con el objetivo de tener almacenado todos los indicadores contables con los que trabajará el sistema.

Es importante resaltar que la base de datos no cuenta con atributos multievaluados ni compuestos, además no existen dependencias parciales de las llaves primarias, ni dependencias transitivas; de lo que se puede inferir, que se encuentra normalizada hasta la tercera forma normal.

2.5 Validación del diseño propuesto

Una métrica es un instrumento de medición que permite evaluar el software al inicio del proceso, que persigue comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado al nivel de proyecto. La aplicación de métricas al diseño de un producto de software constituye un elemento fundamental a la hora de evaluar la calidad del mismo. (32)

Con el fin de desarrollar un diseño robusto y sencillo se realizó la validación del mismo utilizando las métricas Tamaño Operacional de Clase (TOC) y Relaciones entre Clases (RC).

2.5.1 Métrica Tamaño Operacional de Clases (TOC)

Las métricas orientadas a tamaño para una clase Orientada a Objetos se centran en el cálculo de operaciones para una clase individual, y promedian los valores para el sistema Orientado a Objetos en su totalidad. (35) El tamaño general de una clase se determinó empleando la medida: número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.

La métrica TOC posibilita medir los siguientes atributos de calidad:

- **Responsabilidad**: responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.
- **Complejidad de implementación**: grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización**: significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.

En la métrica TOC los atributos de calidad responsabilidad y complejidad de implementación son inversamente proporcionales a la reutilización, lo que puede ser traducido como, mientras mayor sea la responsabilidad y complejidad de implementación de una clase, menor será su nivel de reutilización.

Los valores de umbrales tomados en cuenta para evaluar el diseño propuesto son los siguientes:

Atributo de calidad	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

2.5.1.1 Resultados de la evaluación de la métrica TOC

Como resultado de la evaluación de la métrica TOC se obtuvo lo siguiente:

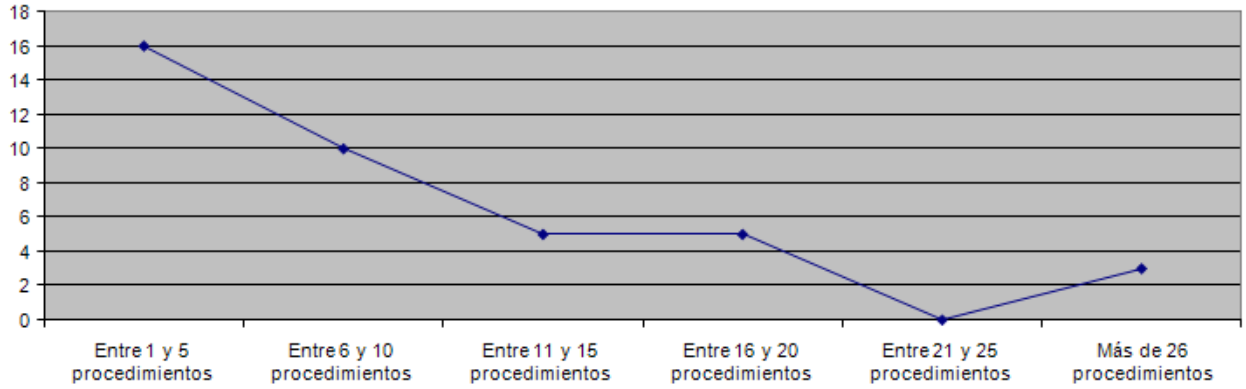


Figura 6: Representación de la cantidad de clases y el número de procedimientos que contienen.

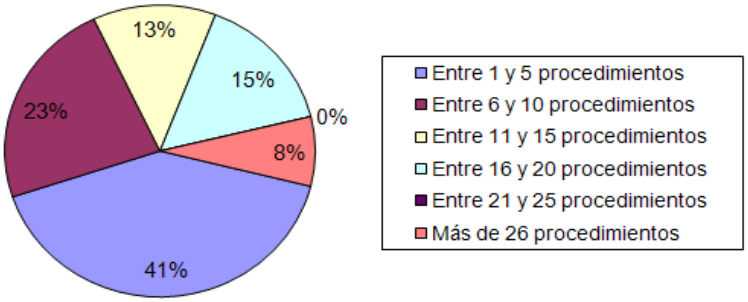


Figura 7: Representación en por ciento de la cantidad de clases y el número de procedimientos que contienen.

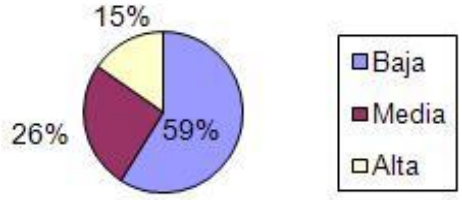


Figura 8: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad.

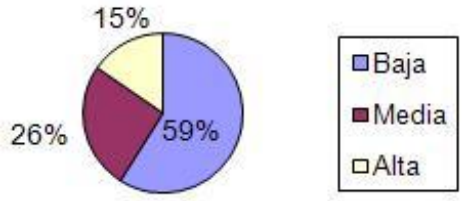


Figura 9: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación.

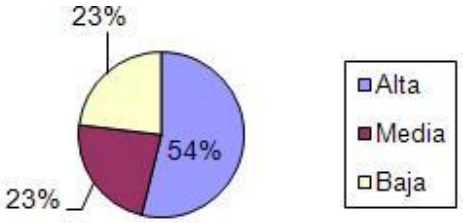


Figura 10: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización.

2.5.1.2 Análisis de los resultados obtenidos en la evaluación de la métrica TOC

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para el módulo Análisis de riesgo de crédito está entre los límites aceptables de calidad. Los atributos de calidad se encuentran en un nivel satisfactorio en la mayoría de las clases; de manera que se puede observar cómo se promueve la reutilización (elemento clave en el proceso de desarrollo de software) y cómo están reducidas la responsabilidad y la complejidad de implementación (59%), lo que promueve el bajo acoplamiento entre las clases y facilitará la implementación y la comprobación al desarrollador.

2.5.2 Métrica Relaciones entre clases (RC)

Se refiere al número de relaciones de uso de una clase. (35) En esta métrica los atributos acoplamiento, complejidad de mantenimiento y cantidad de pruebas son inversamente proporcionales a la reutilización, es decir mientras mayor sea el acoplamiento, complejidad de mantenimiento de una clase y cantidad de pruebas, menor será su nivel de reutilización.

La métrica RC posibilita medir los siguientes atributos de calidad:

- **Acoplamiento**: dependencia o interconexión de una clase o estructura de clase respecto a otras.
- **Complejidad del mantenimiento**: nivel de esfuerzo necesario para sustentar, mejorar o corregir el diseño de software propuesto. Puede influir significativamente en los costes y la planificación del proyecto.
- **Reutilización**: significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.
- **Cantidad de pruebas**: número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto (componente) diseñado.

Los valores de umbrales para dicha métrica son los siguientes:

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
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.

2.5.2.1 Resultados de la evaluación de la métrica RC

Como resultado de la evaluación de la métrica RC se obtuvo lo siguiente:

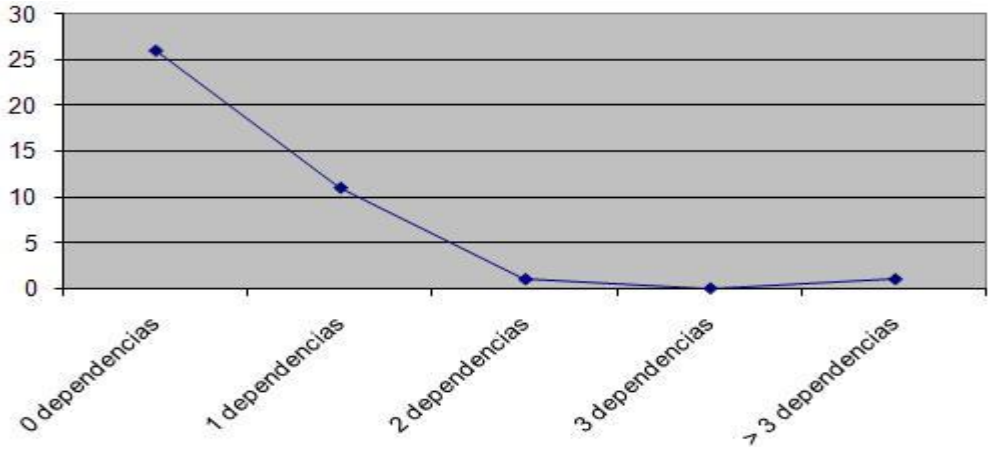


Figura 11: Representación de las asociaciones de uso por cantidad de clases.

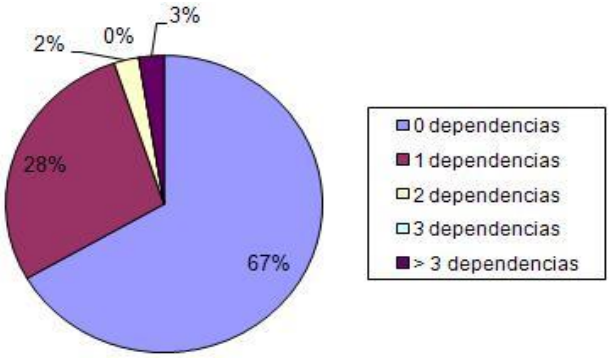


Figura 12: Representación en por ciento de las asociaciones de uso por cantidad de clases.

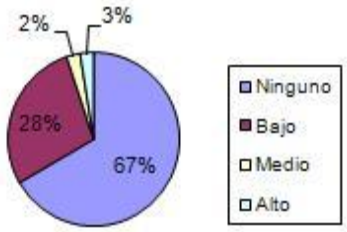


Figura 13: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento.

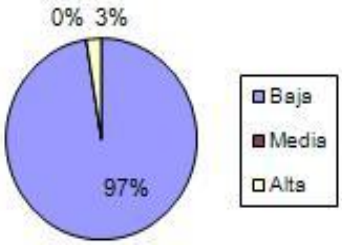


Figura 14: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento.

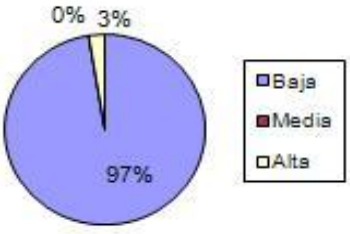


Figura 15: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas.

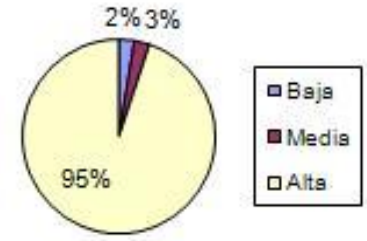


Figura 16: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo reutilización.

2.5.2.2 Análisis de los resultados obtenidos en la evaluación de la métrica RC

De los resultados obtenidos en la evaluación de la métrica RC se puede deducir que el diseño del módulo tiene una calidad aceptable, teniendo en cuenta que el 67% de las clases presentes en el diseño cuentan con una baja cantidad de relaciones de uso. Se puede observar que las clases promueven un bajo nivel de acoplamiento (67%), así como de la complejidad de mantenimiento (97%); igualmente la cantidad de pruebas a realizar representan un bajo por ciento (97%). En consecuencia el grado de reusabilidad es mayor.

En sentido general el resultado de la aplicación de estas métricas demuestra que las clases no se encuentran muy sobrecargadas en responsabilidad, existe además bajo acoplamiento entre las mismas y presentan un alto nivel de reutilización. Indican además que el diseño no es complejo, pues la complejidad de mantenimiento es baja, así como la cantidad de pruebas a realizar. Estos resultados confirman la calidad del diseño.

2.6 Conclusiones parciales

El diseño fue validado por medio de la aplicación de las métricas TOC y RC, donde se evidenció que las clases no se encuentran sobrecargadas en responsabilidad y que presentan un bajo acoplamiento, lo que muestra que la reutilización es alta, al ser esta última inversamente proporcional a los atributos de calidad anteriores.

Las métricas aplicadas permitieron demostrar la calidad del diseño a través de los resultados obtenidos.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

A partir de los resultados obtenidos del diseño, se comenzará la implementación de la solución y después se pasará a realizar las pruebas del sistema. Se mostrarán los artefactos generados en este flujo de trabajo como son: el diagrama de componentes y el modelo de despliegue. Para realizar la validación de la implementación se aplicarán pruebas de caja blanca y caja negra.

3.2 Estándares de codificación

Los estándares de codificación se definen por el equipo de desarrollo para lograr estandarización en la programación del software. Se centran en el aspecto y la estructura física y no en la lógica del programa. El uso de estándares ofrece ventajas ya que, contribuyen a facilitar la lectura, comprensión, mantenimiento y reutilización del código a lo largo del proceso de desarrollo de un software. (36)

3.3 Convenciones de nomenclatura

Las convenciones generales de nomenclatura explican la elección de los nombres más adecuados para todos los identificadores en cualquier lenguaje de programación.

En la implementación del módulo Análisis de riesgo de crédito la nomenclatura de las clases está definida por la utilización de la notación Pascal Casing.

La notación Pascal Casing define que los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas iniciando cada palabra con letra mayúscula, según lo indicado por (36). Ejemplo: AnalisisRiesgoManager. En este caso el nombre de la clase está compuesto por tres palabras iniciadas cada una con letra mayúscula.

También se tomó en cuenta para el nombrado de las clases el tipo que esta posee, o sea el rol que ellas desempeñan en el sistema. A continuación se presentan las nomenclaturas organizadas por los paquetes a los que pertenecen las clases.

controller: las clases incluidas en este paquete, después del nombre se le incorpora el nombre del controlador de Spring del cual hereda (NombreClaseNombreControladorClases). Ejemplo: RegistrarInformacionFinancieraMultiActionController.

command: las clases que se ubican dentro de este paquete se nombran con el nombre de la clase más la palabra Command (NombreClaseCommand). Ejemplo: InformacionFinancieraCommand.
validator: en este paquete la nomenclatura de las clases está determinada por el nombre de estas más la palabra Validator (NombreClaseValidator). Ejemplo: AnalisisRiesgoValidator.
propertyEditor: las clases que se encuentran dentro de este paquete después del nombre se le agrega la palabra PropertyEditor (NombreClasePropertyEditor). Ejemplo: RegistrarPropertyEditor.
flowHandler: en este paquete a las clases después del nombre se les coloca la palabra FlowHandler. El nombre responde a lo que realiza el flujo que se quiere personalizar. Ejemplo: RegistrarInformacionFinancieraFlowHandler.
serviceFlow: al nombre de las clases que están dentro de dicho paquete se le agrega la palabra Action o MultiAction en dependencia de cuál de las dos clases herede (NombreClaseAction|MultiAction). Ejemplo: RegistrarInformacionFinancieraMultiAction.
facade: las clases incluidas en este paquete, después del nombre del módulo en cuestión se le agrega la palabra Facade y en el caso del subpaquete impl tendrán el mismo nombre de la interfaz que implemente más la palabra Impl (nombreDelModuloFacade, NombreDelModuloFacadeImpl). Ejemplo: AnalisisRiesgoFacade, AnalisisRiesgoFacadeImpl.
manager: las clases que se encuentran en este paquete después del nombre del negocio llevan la palabra Manager (NombreDelNegocioManager). En el caso del subpaquete impl tendrán el mismo nombre de la interfaz que implementa más la palabra Impl (NombreDelNegocioManagerImpl). Ejemplo: GestionarAnalisisRiesgoManager, GestionarAnalisisRiesgoManagerImpl.
dao: las clases incluidas en este paquete, después del nombre de la entidad a la que responden se le incorpora la abreviatura DAO (NombreEntidadDAO) y en el caso del subpaquete impl tendrán el mismo nombre de la interfaz que implementa más la palabra Impl (NombreEntidadDAOImpl). Ejemplo: InformacionFinancieraDAO, InformacionFinancieraDAOImpl.

De manera general el nombre de los métodos y los atributos de las clases se escriben con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará notación Camel-Casing, que es muy similar a Pascal Casing con la excepción de que la letra inicial debe estar en minúscula. Ejemplo: registrarInformacionFinanciera, este nombre de método está compuesto por dos palabras, la primera todo en minúsculas y la segunda iniciando con letra mayúscula. Lo mismo se aplica a los nombres de ficheros de código JavaScript y sus funciones y variables internas.

Los comentarios deben ser lo más claros y precisos posibles de forma tal que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se debe comentar para lograr una mejor comprensión del código.

3.4 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente y ejecutables. Describe también cómo se organizan los componentes de acuerdo a los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado y cómo dependen los componentes unos de otros, como indica (14).

3.4.1 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Los componentes son un conjunto de funcionalidades comunes que serán reutilizados por otros módulos del sistema. En algunas ocasiones se comportarán como módulos visuales en el sistema, y en otras ocasiones solamente recogerán funcionalidades del negocio. (37)

El diagrama de componentes que se muestra a continuación se ha elaborado de forma tal que muestre las relaciones existentes entre el módulo Análisis de riesgo de crédito y el resto de componentes con los que se relaciona dentro del sistema.

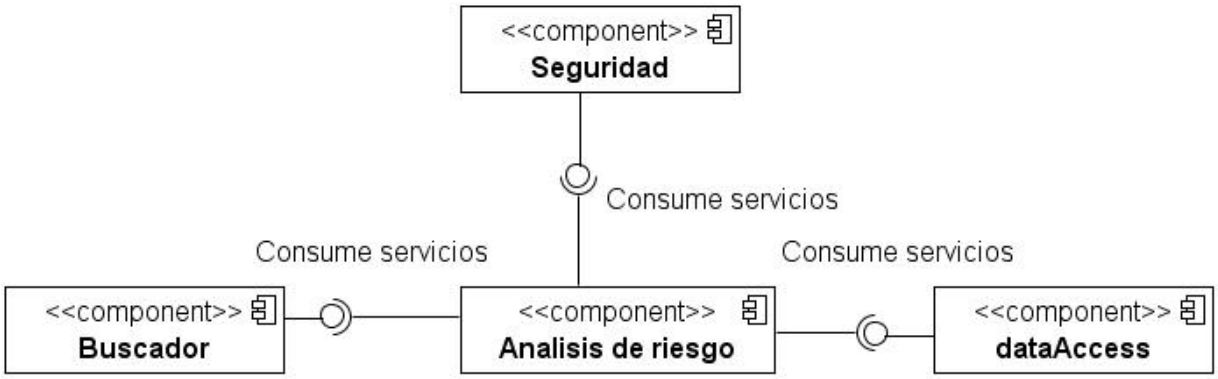


Figura 17: Modelo de componentes correspondiente al módulo Análisis de riesgo de crédito.

A continuación se explican de manera general cada uno de los componentes.

Seguridad: como su nombre lo indica es el encargado de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos, en dependencia del usuario que la

realice, de aquí que sea necesaria la comunicación con el mismo para garantizar la seguridad del módulo para el análisis de riesgo de crédito.

Buscador: ofrece un conjunto de clases que constituyen el motor de búsqueda, presentando una interfaz gráfica mediante la cual se solicitan los diferentes conceptos en dependencia del módulo donde se emplee. Para la correcta gestión de la información en el módulo Análisis de riesgo de crédito se hace necesaria la búsqueda de informaciones financieras, indicadores contables y dictámenes de riesgo.

Por otro lado el componente **dataAccess** tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos y por ende todos los subsistemas dependen de él para poder realizar las funcionalidades que engloban.

3.4.2 Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Muestra la configuración de los elementos de hardware (nodos) y artefactos del software que se trazan en esos nodos, representando la distribución física del sistema en términos de cómo se distribuirán las funcionalidades entre los nodos. Cada nodo representa un recurso de cómputo, siendo estos procesadores o dispositivos hardware que se necesitarán para el despliegue del sistema, según lo indicado por (14). La figura 18 muestra el diagrama de despliegue que se utilizará en el BNC.

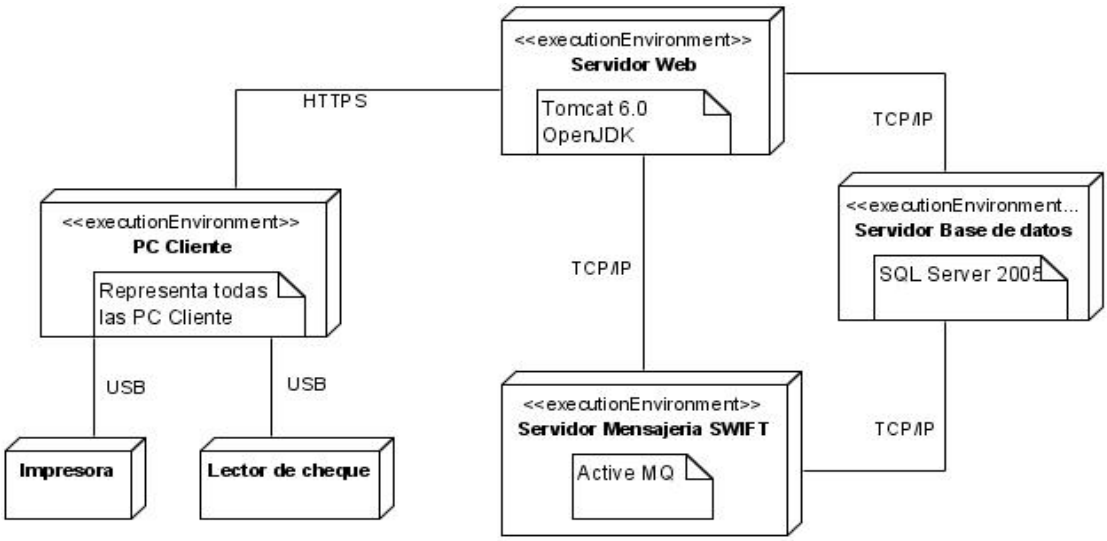


Figura 18: Modelo de despliegue para el BNC. (13)

3.5 Descripción de las clases principales y sus funcionalidades

Teniendo en cuenta el diseño de las clases correspondientes al módulo Análisis de riesgo de crédito realizado anteriormente, a continuación serán descritos los atributos y métodos de algunas clases que son muy importantes para el módulo desde el punto de vista funcional.

Tabla 1: Descripción de la clase RegistrarInformacionFinancieraMultiAction.

Nombre: RegistrarInformacionFinancieraMultiAction	
Tipo de clase: Controladora	
Atributo	**Tipo**
analisisRiesgoFacade	AnalisisRiesgoFacade
globalFacade	GlobalFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
inicializarFlujo(RequestContext context)	Permite obtener el objeto Información financiera que servirá de modelo a la vista.
registrarInformacionFinanciera (RequestContext contex)	Es donde se prepara el objeto de la clase información financiera que ha sido entrada, para luego ser registrada en el sistema.

Tabla 2: Descripción de la clase GestionarAnalisisRiesgoManagerImpl.

Nombre: GestionarAnalisisRiesgoManagerImpl	
Tipo de clase: Controladora	
Atributo	**Tipo**
globalFacade	GlobalFacade
informacionFinancieraDAO	InformacionFinancieraDAO
balanceGeneralDAO	BalanceGeneralDAO
estadoResultadoDAO	EstadoResultadoDAO

parametroBalanceGeneralDAO	ParametroBalanceGeneralDAO
parametroEstadoResultadoDAO	ParametroEstadoResultadoDAO
indicadoresDAO	IndicadoresDAO
dictamenDAO	DictamenDAO
Para cada responsabilidad:	
Nombre:	**Descripción**:
registrarInformacionFinanciera(InformacionFinanciera infin)	Registra una información financiera.
listarInformacionFinanciera()	Obtiene una lista con todos los objetos información financiera existentes.
buscarIFPorId()	Obtiene una información financiera dado el identificador de la misma.
eliminarIF()	Elimina una información financiera existente.
registrarIndicadoresContables (Indicadores indicador)	Registra un indicador.
priorizarIndicadoresContables()	Permite dar prioridad a los indicadores contables.
listarIndicadores()	Obtiene una lista con todos los indicadores existentes.
buscarIndicadorPorId()	Obtiene un indicador dado el identificador del mismo.
eliminarIndicador()	Elimina un indicador existente.
crearDictamenFinal()	Permite registrar un dictamen final.
crearString()	En este método se crea el dictamen final que se le muestra al usuario.
listarDictamenFinal()	Obtiene una lista con todos los dictámenes

	existentes.
buscarDFPorId()	Obtiene un dictamen dado el identificador del mismo.
eliminarDictamen()	Elimina un indicador existente.
obtenerEstadoSistema()	Obtiene el estado en el que se encuentra el sistema.
obtenerParametrosBalanceGeneral()	Obtiene los parámetros del balance general.
obtenerParametrosEstadoResultado()	Obtiene los parámetros del estado de resultados.

3.6 Aspectos principales de la implementación

Utilización de Spring Web Flow Framework

A lo largo del desarrollo del presente trabajo, se ha hecho énfasis en la utilización de Spring Web Flow por las facilidades que brinda en el desarrollo del software, principalmente en escenarios donde existen flujos complejos de vistas e información. Para el desarrollo del módulo Análisis de riesgo de crédito se emplea este framework en todas las funcionalidades que su módulo comprende. Se hace uso de Spring Web Flow mayormente, debido a la complejidad que posee el módulo. A continuación se describe el funcionamiento del mismo.

El principal aspecto de este módulo es la existencia de un flujo capaz de redireccionar a diferentes estados de vista y trabajar con el mismo objeto en diferentes fases. Por cada funcionalidad (registrar, actualizar y consultar) se creó un flujo que inicia con la siguiente declaración:

```
<var name="informacionFinancieraCommand"
    class="cu.uci.finixubnc.credito.analisisriesgo.web.webflow.command.InformacionFinancieraCommand"/>

<on-start>
    <evaluate expression="registrarInformacionFinancieraMultiAction.inicializarFlujo"/>
</on-start>

<view-state id="registrarInformacionFinanciera" model="informacionFinancieraCommand"
    view="registrarInformacionFinanciera">
    <transition on="aceptar" to="registrar" />
    <transition on="cancelar" to="end" />
</view-state>

<action-state id="registrar">
    <evaluate
        expression="registrarInformacionFinancieraMultiAction.registrarInformacionFinanciera"/>
    <transition on="success" to="registrarInformacionFinanciera" />
</action-state>

<end-state id="end"
    view="externalRedirect:servletRelative:../../common/home.htm">
</end-state>
```

Figura 19: Declaración del flujo para el CU Registrar información financiera.

En este caso se muestra y explica el flujo para registrar una información financiera. Se declara la variable **informacionFinancieraCommand** que representa el modelo del que se estará haciendo uso en la vista, el cual será modificado en dependencia de la funcionalidad que se esté realizando con la llamada al método **inicializarFlujo (RequestContext context)** en la etiqueta **on-start**.

Todas las funcionalidades presentes en el flujo son implementadas en la clase **RegistrarInformacionFinancieraMultiAction**, la cual se describió con anterioridad. Las funcionalidades declaradas en el flujo no necesitan declararse con los parámetros correspondientes, debido a que si reciben el **RequestContext** como único parámetro, Spring Web Flow es capaz de reconocerlo.

Posteriormente se prepara la vista que va a responder a la petición realizada mediante la etiqueta **view-state**. Una vez dentro se ejecutan los estados de transición, son los encargados de decidir a qué estado de vista se moverá el flujo en dependencia de la variable **to**.

Si desde la página que se muestra al usuario se ejecuta el evento **aceptar**, se desencadenarán una serie de estados de acción y de decisión que manejarán los pasos para registrar la información

financiera, que culminarán en un estado de finalización **end-state**, donde está definida la página cliente con la que se dará por terminada la funcionalidad. Si por el contrario se ejecutara el evento **cancelar**, el estado de transición iría directamente al estado final.

3.7 Validación de la implementación del módulo

La calidad de un producto de software se ha convertido en un factor indispensable de las grandes organizaciones debido a su fuerte impacto en la competitividad de las empresas. Durante el proceso de desarrollo de software las posibilidades de errores son múltiples por lo que se hace necesario detectar a tiempo las imperfecciones y proporcionar una visión de la calidad de los procesos asociados. El objetivo fundamental que rige esta etapa es determinar mediante las pruebas, cómo y en qué medida el módulo Análisis de riesgo de crédito cumple con las expectativas del cliente.

3.7.1 Pruebas de Software

Las pruebas constituyen una etapa imprescindible durante el proceso de desarrollo del software. Su objetivo principal es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que este pudiera tener. (38) Las pruebas permiten detectar y corregir el máximo de errores posibles antes de la entrega al cliente del software desarrollado, por lo que el éxito de las mismas puede mejorar la apreciación de calidad del usuario final y lograr su satisfacción.

Las pruebas que se le realizaron al módulo corresponden al nivel de Unidad, las cuales están enfocadas al código fuente de los componentes para verificar todos los flujos de control, probando de manera individual las partes del sistema que han sido desarrolladas. Adecuadas a este nivel se aplicarán los métodos de pruebas de Caja Blanca; para analizar la lógica interna del programa y Caja Negra con el objetivo de verificar que las funciones son operativas a través de la interfaz del software, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, manteniendo la integridad de la información externa.

Un método de prueba es un enfoque sistemático, independiente del nivel en que se enmarque la prueba, que ayuda a encontrar buenos conjuntos de casos de prueba para detectar diferentes tipos de errores. Los casos de prueba especifican una forma de probar el componente. (39)

3.7.1.1 Aplicación de la prueba de Caja Blanca o Estructural

Esta prueba consiste específicamente en diseñar los Casos de prueba atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento. (40)

Para realizar esta prueba de calidad se utilizó el framework JUnit, el cual se puede integrar al IDE de desarrollo Eclipse. Según lo indicado por (41), este framework permite la automatización de las pruebas de aplicaciones en Java, consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba y ejecutarlos automáticamente. Además mejora el diseño de implementación, haciéndolo flexible y testeable.

Para realizar las pruebas de caja blanca con JUnit se toma como objeto de estudio el método existeDR() (devuelve true si existe un dictamen asignado a una información financiera, dado su identificador) que se encuentra en la clase CargarDatosAnalisisRiesgoMultiActionController, el cual se muestra a continuación.

```
public boolean existeDR(HttpServletRequest request,
        HttpServletResponse response) {
    String a="False";
    Integer idIF = Integer.parseInt(request.getParameter("idIF"));
    InformacionFinanciera inf = analisisRiesgoFacade.buscarIFPorId(idIF);
    List<DictamenFinal> listaDR = analisisRiesgoFacade.listarDictamenFinal();

    for (int i = 0; i < listaDR.size(); i++) {
        if(listaDR.get(i).getInformacionFinanciera().getIdInformacionFinanciera()
                == inf.getIdInformacionFinanciera())
            a ="True";
        if(a.equals("True"))
            break;
    }
    try {
        ResponseUtil.escribirDatosEnElResponse(response, a);
    } catch (IOException e) {
        e.printStackTrace();
    };
    return true;
}
```

Figura 20: Método testeado por el framework JUnit.

Para realizarles las pruebas a dicho método primeramente se crea una clase la cual debe extender de TestCase, en este caso dicha clase se nombra TestJUnit en la cual se realizan una serie de pasos los cuales se muestran y describen a continuación.

```
public class TestJUnit extends TestCase {

    CargarDatosAnalisisRiesgoMultiActionController cargarDatos;
    private MockControl controlHttpServlet;
    private HttpServletRequest mockhttpServletRequest;
    private MockControl controlHttpResponse;
    private HttpServletResponse mockhttpServletResponse;
    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;

    protected void setUp()throws Exception {
        cargarDatos = new CargarDatosAnalisisRiesgoMultiActionController();
        ApplicationContext context = new ClassPathXmlApplicationContext(
        "classpath:cu/uci/finixubnc/credito/JUnit/analisisriesgo-context.xml");
        GestionarAnalisisRiesgoManagerImpl manager = (GestionarAnalisisRiesgoManagerImpl)
            context.getBean("gestionarAnalisisRiesgoManager");
        AnalisisRiesgoFacadeImpl facade = new AnalisisRiesgoFacadeImpl();
        facade.setGestionarAnalisisRiesgoManager(manager);
        cargarDatos.setAnalisisRiesgoFacade(facade);

        controlHttpServlet= MockControl.createControl(HttpServletRequest.class);
        mockhttpServletRequest=(HttpServletRequest) controlHttpServlet.getMock();
        controlHttpResponse= MockControl.createControl(HttpServletResponse.class);
        mockhttpServletResponse=(HttpServletResponse) controlHttpResponse.getMock();
        controlHttpSession= MockControl.createControl(HttpSession.class);
        mockHttpSession = (HttpSession) controlHttpSession.getMock();
        super.setUp();
    }
```

Figura 21: Clase TestJUnit para realizar la prueba.

Primeramente se crea un objeto de la clase en la cual se encuentra el método que se va a testear, en este caso el objeto es cargarDatos de tipo CargarDatosMultiActionController. Después se crean dos simulacros, uno de HttpServletRequest y otro de HttpServletResponse que son los parámetros que se les pasa al método que se quiere probar, además de sus respectivos MockControl encargados de su control. Posteriormente en el método setUp () se inicializan todos los atributos anteriormente creados, además de un contexto, el cual contiene todos los beans de las clases necesarias para la ejecución del procedimiento que se va a probar.

```
protected void tearDown() {
    controlHttpServlet.verify();
}

public void testExisteDR() throws Exception{
    mockhttpServletRequest.getParameter("idIF");
    controlHttpServlet.setReturnValue("34");
    controlHttpServlet.replay();
    assertTrue(cargarDatos.existeDR(mockhttpServletRequest,mockhttpServletResponse));
}
```

Figura 22: Métodos tearDown() y testExisteDR() de la clase TestJUnit.

En el método tearDown es donde se verifican si las llamadas a los métodos se realizaron correctamente, es el encargado de informar la existencia de los errores en la realización de las pruebas. Las verificaciones en este método se realizan de forma automática para todos los test definidos en la clase.

Una vez definido el tearDown se crea un método, el cual contendrá los elementos para realizar el test. En dicho procedimiento creado (testExisteDR()), se especifica cómo son recibidos los parámetros y cuál es el identificador de cada uno, en el método que será sometido a prueba. Posteriormente mediante assertTrue() se ejecuta el procedimiento que está siendo analizado y se espera que el framework termine el análisis. Según el resultado del mismo, JUnit muestra una ventana en correspondencia con este: si es satisfactorio, mediante una línea de color verde, en caso contrario de color rojo. A continuación se muestra el caso favorable para dicha prueba.

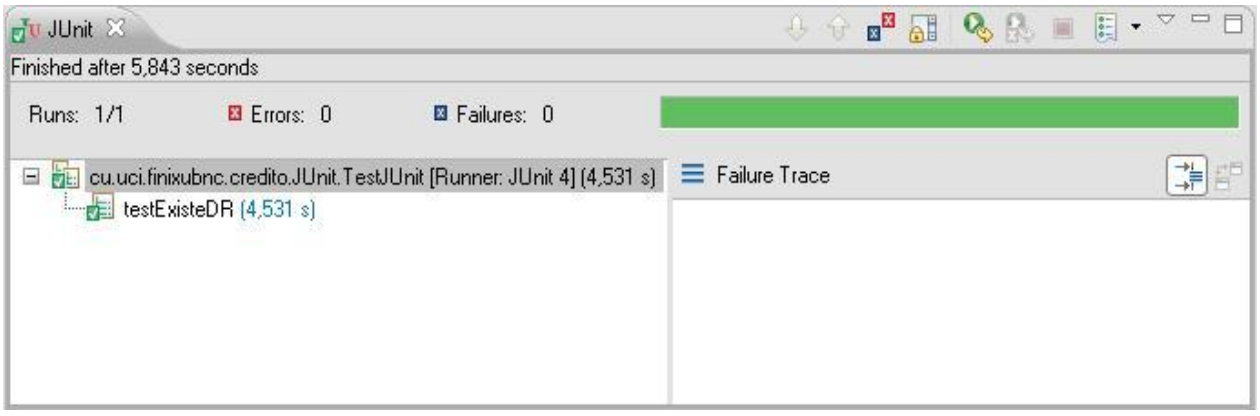


Figura 23: Resultado de la prueba de caja blanca aplicado al método existeDR().

3.7.1.2 Aplicación de la prueba de Caja Negra o Funcional

Los requisitos de software se comprueban utilizando técnicas de diseño de casos de prueba de caja negra. Con estos diseños, se intenta encontrar el mayor número de errores con la menor cantidad de esfuerzo y tiempo. Un caso de prueba no es más que un conjunto de condiciones bajo las cuales se determina si un requisito es parcial o completamente satisfactorio. Su propósito es especificar una forma de probar el sistema que incluya las entradas, los resultados esperados y las condiciones bajo las que ha de probarse. (40) Las pruebas de caja negra se realizan sobre la interfaz de usuario e intentan descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca.

Por cada requisito funcional del módulo se realizó un Diseño de Casos de Prueba (DCP, en lo adelante). Estos diseños fueron aplicados por los revisores técnicos del proyecto SAGEB.

A continuación se presenta el DCP para el requisito funcional Registrar información financiera, los DCP de los restantes requisitos funcionales se encuentran en el Anexo 5: Diseños de Casos de Prueba del módulo Análisis de riesgo de crédito.

1. **Descripción General.**
   Se registra una nueva información financiera en el sistema.

2. **Condiciones de Ejecución.**
   El usuario debe estar autenticado en el sistema.

3. Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Registrar información financiera.	EC 1.1: Registrar con éxito.	1. El usuario escribe en el navegador la dirección para acceder a la aplicación. 2. Da clic en el botón Aceptar. 3. Selecciona el subsistema Créditos. 4. Elige el módulo Información financiera. 5. Escoge la opción Registrar. 6. El sistema muestra la interfaz para registrar una información financiera. 7. El usuario entra los datos necesarios para realizar esta operación. 8. Selecciona la opción Aceptar. 9. El sistema muestra el mensaje: Operación realizada satisfactoriamente. 10. El sistema redirecciona a la página inicial del módulo. 11. Finaliza la operación.

	EC 1.2: Datos incorrectos.	1. El usuario escribe en el navegador la dirección para acceder a la aplicación. 2. Da clic en el botón Aceptar. 3. Selecciona el subsistema Créditos. 4. Elige el módulo Información financiera. 5. Escoge la opción Registrar. 6. El sistema muestra la interfaz para registrar una información financiera. 7. El usuario introduce los datos necesarios para registrar. 8. Selecciona la opción Aceptar. 9. El sistema valida los datos y muestra el mensaje según el error ocurrido. 10. El usuario corrige los datos. 11. Selecciona la opción Aceptar. 12. El sistema muestra el mensaje: Operación realizada satisfactoriamente. 13. El sistema redirecciona a la página de inicio del módulo. 14. Finaliza la operación.
	EC 1.3: Cancelar	1. El usuario escribe en el navegador la dirección para acceder a la aplicación. 2. Da clic en el botón Aceptar. 3. Selecciona el subsistema Créditos. 4. Escoge el módulo Información financiera. 5. Elige la opción Registrar. 6. El sistema muestra la interfaz para registrar una información financiera. 7. El usuario selecciona la opción Cancelar. 8. El sistema muestra el mensaje: ¿Está seguro que desea cancelar la operación? 9. El usuario selecciona Sí para cancelar. 12. El sistema redirecciona a la página inicial del módulo. 13. Finaliza la operación.

4. Descripción de variable.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
[1]	Fecha actual	Calendario	No	Seleccionar la fecha de la información financiera actual.
[2]	Fecha anterior	Calendario	No	Seleccionar la fecha de la información financiera anterior.
[3]	Cliente	Lista desplegable	No	Se selecciona el cliente al cual pertenece la información financiera.
[4]	Fecha actual	Campo numérico.	No	Se escribe el valor que van tomando los parámetros del balance general y el estado de resultado en la fecha presente.
[5]	Fecha anterior	Campo numérico.	No	Se escribe el valor que van tomando los parámetros del balance general y el estado de resultado en la fecha anterior.
[6]	Variación	Campo numérico.	No	Se calcula automáticamente.

5. Matriz de Datos.

5.1 SC 1 Registrar información financiera.

Escenario	Fecha actual	Fecha anterior	Cliente	Fecha actual	Fecha anterior	Variación	Respuesta del Sistema	Resultado de la Prueba
Registrar con éxito.	V (15/05/2012)	V (15/05/2011)	V (00006 - CIMTEC S.A.)	V (300)	V (200)	V (100)	Se registra la información financiera satisfactoriamente.	Satisfactorio.
Datos incorrectos.	I (Martes)	V (15/05/2011)	V (00006 - CIMTEC S.A.)	V (300)	V (200)	V (100)	El sistema muestra el mensaje: El valor especificado no es válido.	Satisfactorio.

Escenario	Fecha actual	Fecha anterior	Cliente	Fecha actual	Fecha anterior	Variación	Respuesta del Sistema	Resultado de la Prueba
	V (15/05/2012)	I (Martes)	V (00006 - CIMTEC S.A.)	V (300)	V (200)	V (100)	El sistema muestra el mensaje: El valor especificado no es válido.	Satisfactorio.
	V(15/05/2012)	V(15/05/2011)	I(/esc)	V(300)	V(200)	V(100)	El sistema muestra el mensaje: El valor especificado no es válido.	Satisfactorio
	V(15/05/2012)	V(15/05/2011)	V (00006 - CIMTEC S.A.)	I(dos)	V(200)	V(100)	El sistema muestra el mensaje: El valor especificado no es válido.	Satisfactorio.
	V(15/05/2012)	V(15/05/2011)	V (00006 - CIMTEC S.A.)	V(300)	I(dos)	V(100)	El sistema muestra el mensaje: El valor especificado no es válido.	Satisfactorio.
	V(15/05/2012)	V(15/05/2011)	V (00006 - CIMTEC S.A.)	V(300)	V (200)	NA	El sistema calcula automáticamente el valor de la variación.	Satisfactorio.
Cancelar	NA	NA	NA	NA	NA	NA	El sistema muestra el mensaje: ¿Está seguro que desea cancelar la	Satisfactorio

Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.]

Después de realizadas las pruebas, en la primera iteración se detectaron 5 no conformidades, las cuales fueron resueltas inmediatamente. Para la segunda iteración se detectaron 2 no conformidades

resolviéndose para una tercera iteración, señalándose la obtención de resultados válidos que proceden con satisfacción.

Después de realizadas las pruebas, la solución propuesta fue presentada al cliente, el cual demostró buen nivel de satisfacción validando la propuesta de solución para realizar el proceso de análisis de riesgo de crédito en el BNC, evidenciándose a través de la carta de aceptación. Para más detalles ver Anexo 6 Carta de aceptación del BNC.

3.8 Conclusiones parciales

Las pruebas de software aplicadas para la validación de la implementación se efectuaron mediante casos de prueba arrojando resultados satisfactorios que complementan la calidad de la solución.

Se puede afirmar que el módulo Análisis de riesgo de crédito cumple desde el punto de vista funcional con los requerimientos y expectativas del cliente.

CONCLUSIONES

Una vez terminado el presente trabajo de diploma se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos:

- Se detectó que ninguno de los sistemas informáticos estudiados cumplen con los requerimientos necesarios para realizar el proceso de análisis de riesgo en el BNC.
- El diseño y la implementación del módulo Análisis de riesgo de crédito del sistema Quarxo facilitó llevar a cabo el proceso de análisis de riesgo de crédito en el BNC.
- Se evaluó el módulo Análisis de riesgo de crédito a través de métricas y pruebas aplicadas al sistema, las cuales arrojaron resultados favorables posibilitando dar cumplimiento a las funcionalidades previstas para el mismo.

La solución resulta novedosa y su importancia radica en la realización de los procesos referentes al análisis de riesgo de crédito en el BNC, a través de un módulo que funciona rápido y correctamente, permitiendo el ahorro de tiempo y recursos, así como el control eficiente de los cambios y la información manipulada.

RECOMENDACIONES

Teniendo en cuenta la investigación realizada en el presente trabajo de diploma, se recomienda:

- Realizar funcionalidades que permitan registrar la información financiera de forma automática a partir de tablas Excel.
- Realizar un sistema de reportes para imprimir los resultados obtenidos en el análisis de riesgo.

REFERENCIAS BIBLIOGRÁFICAS

1. **FOGACOOP.** Riesgo de Crédito. [En línea] [Citado el: 28 de Noviembre de 2011.] http://www.fogacoop.gov.co/documentos/PRESENTACI%D3N%20RIESGO%20DE%20CR%C9DITO.ppt.

2. **Círculo de Periodistas Cubanos contra el Terrorismo.** Cubadebate, contra el terrorismo mediático. [En línea] 2012. [Citado el: 14 de Enero de 2012.] http://www.cubadebate.cu/wp-content/uploads/2011/05/tabloide_debate_lineamientos.pdf.

3. **García Romero, Yaniuska y Fernández Miranda, Denise.** *Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. Análisis y Diseño del módulo Análisis de Riesgo de Crédito del proyecto SAGEB.* 2010.

4. **García Suárez, Lic. Arlenis.** *Evolución del sistema bancario y financiero cubano a partir de 1995.* 2005.

5. **Toca Ramos, Lázaro Alberto.** *Estudio de la Contabilidad General.* La Habana : Félix Varela, 2010.

6. **Gestiopolis.** Aspectos básicos del análisis de riesgo. [En línea] [Citado el: 29 de Noviembre de 2011.] http://www.gestiopolis.com/recursos/documentos/fulldocs/fin/aspanalisiscreditos.htm.

7. **Banca Afirme.** Administración Integral de riesgos. *El banco de hoy.* [En línea] [Citado el: 2 de Diciembre de 2011.] http://www.afirme.com.mx/Portal/VisualizadorContenido.do?archivoId=1573&menu=1.

8. **Buniak, Leonardo.** Gestión de Riesgo para Instituciones Financieras. [En línea] Diciembre de 2011. http://www.buniak.com/negocio.php?id_seccion=8&id_documento=166.

9. **López Rodríguez, Ing. Yoan Antonio, Saavedra Darias, Ing. Mavi y Osorio Turruelles, Ing. Yoel.** Sistema para el análisis de riesgo de créditos (CREDIRI). [En línea] [Citado el: 12 de Junio de 2012.] http://uciencia.uci.cu/es/node/1056.

10. Información General de @Risk. [En línea] http://www.software-shop.com/in.php?mod=ver_producto&prdID=288.

11. Scalar Consulting. [En línea] [Citado el: 15 de Noviembre de 2011.] http://www.grupoescalar.com/software/riesgo_credito_scoring.htm.

12. **Piñero Pérez, Pedro.** *Metodologías Ágiles y Robustas.* 2009.

13. **Iglesias Chaviano, Adolfo Miguel.** *Documento de la Arquitectura, Modernización Sistema del Banco Nacional de Cuba.*

14. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* 2000.

15. **Universitat Jaume I de Castellón Grupo de Ingeniería del Diseño .** *Modelo informal basado en IDEF4 para la representación de diseños basados en el esquema FBS.* Castellón : s.n., 2012.

16. **Larman, Craig.** *UML y Patrones.* s.l. : Prentice Hall. 1999. 970-17-0261-1.

17. **Desarrolloweb.com, Equipo.** ¿Qué es Java? [En línea] 1999. [Citado el: 24 de Noviembre de 2011.] http://www.desarrolloweb.com/articulos/497.php.

18. **Oracle Corporation.** Sun MicroSystems. [En línea] [Citado el: 28 de Noviembre de 2011.] http://java.sun.com/javaee/5/docs/tutorial/doc/.

19. **http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html., 1. Manual de XHTML. [En línea] [Citado el: 24 de Noviembre de 2011.].** Manual de XHTML. [En línea] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.

20. **Rodríguez, Jose Antonio.** MANUAL DE JAVASCRIPT . [En línea]

21. Codebox. Glosario. [En línea] http://www.codebox.es/glosario.

22. **Seth, Seth Ladd y Keith, Donald.** *Expert Spring MVC and Web Flows.* 2006.

23. **Peak, Patrick y Heudecker, Nick.** Hibernate Quickly.

24. **Russell, Matthew A.** *Dojo The Definitive Guide.* Gravenstein Highway North : O'Reilly Inc., 2008. ISBN: 978-0-596-514648-2.

25. **Scribd.** Scribd. [En línea] [Citado el: 28 de Noviembre de 2011.] http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

26. Visual Paradigm. [En línea] Diciembre de 2011. [Citado el: 28 de Noviembre de 2012.] http://www.visual-paradigm.com.

27. Control de versiones con Subversion. [En línea] http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html.

28. **Eclipse Foundation.** Eclipsepedia. [En línea] [Citado el: 2 de Diciembre de 2011.] http://wiki.eclipse.org/Main_Page.

29. Apache Tomcat. [En línea] http://tomcat.apache.org/.

30. **Microsoft Corporation.** Microsoft SQL Server. [En línea] [Citado el: 28 de Noviembre de 2011.] http://www.microsoft.com/sqlserver/2005/en/us/product-information.aspx.

31. **Desarrollo Web.** Usabilidad y arquitectura del software. [En línea] 2011. [Citado el: 15 de Enero de 2012.] http://www.desarrolloweb.com/articulos/1622.php.

32. **S. Pressman, Roger.** *Ingeniería de Software un enfoque práctico.* Quinta Edición. Madrid : s.n., 2001.

33. **KENDALL, KENNETH E. y KENDALL, JULIE E.** *Análisis y diseño de sistemas.* Sexta Edición. México : PEARSON EDUCACIÓN, 2005. págs. 693-694. ISBN 970-26-0577-6.

34. Definición de modelo de datos. [En línea] 2012. http://definicion.de/modelo-de-datos/.

35. **Negro, Ing. Pablo Ariel.** Umbrales para métricas orientada a objetos. [En línea] 2008. http://caeti.uai.edu.ar/archivos/271_TESIS.PDF.

36. **Smania, Sofía Duarte y Noelia.** Tecnologías en Desarrollo de Software IDE - UTN. [En línea] 20 de Mayo de 2008. [Citado el: 5 de Junio de 2012.] http://subversion.assembla.com/svn/net-utn2008/trunk/TPs%201/08.309.TP1.Smania-Duarte.Convenciones/.

37. **Guerrero, Luis A.** Taller de UML. s.l. : Universidad de Chile.

38. **S. Pressman, Roger.** Ingeniería del Software, un enfoque práctico. Cuarta Edición. McGraw-Hill : s.n., 1997.

39. **Ramírez, Jaime.** Unidad de Programación. Métodos de Prueba del Software. [En línea] 2012. [Citado el: 28 de Mayo de 2012.] http://lml.ls.fi.upm.es/.

40. **S. Pressman, Roger.** *Ingeniería de Software un enfoque práctico.* Sexta Edición. 2001.

41. **Massol, Vincent y Husted, Ted.** *JUnit in Action.* s.l. : Manning Publications Co., 2004. ISBN 1-930110-99-5.

42. **Círculo de Periodistas Cubanos contra el Terrorismo.** Juventud Rebelde. Diario de la Juventud Cubana. [En línea] 29 de Noviembre de 2011. http://www.juventudrebelde.cu/cuba/2011-11-29/creditos-que-dinamizaran-la-economia/.

GLOSARIO DE TÉRMINOS

API: Application Programming Interface (Interfaz de Programación de Aplicaciones en español) es un conjunto de funciones y procedimientos que ofrece determinada biblioteca para ser utilizada por otro software.

Balance general: es el estado que muestra en unidades monetarias la situación financiera de una empresa o entidad económica en un momento determinado. Comprende información clasificada y agrupada en tres grupos principales: activos, pasivos y capital. Dentro de los activos se encuentra todo lo que posee valor para la empresa, entre ellos: el dinero en caja y en bancos, las cuentas por cobrar a los clientes. Por su parte lo pasivo es todo lo que la empresa debe y se clasifica según el orden de exigibilidad en pasivos corrientes, pasivos a largo plazo y otros pasivos. El valor de lo que pertenece al empresario a la fecha de realización del balance se le conoce como Patrimonio y se clasifica en capital, utilidades retenidas y utilidades del período anterior. El patrimonio es la diferencia entre el activo y el pasivo.

Dictamen final: documento elaborado por el Analista de Riesgo, constituye la valoración final de este sobre el proceso de análisis de riesgo de crédito.

Información financiera: conjunto de datos que se utilizan para conocer el patrimonio o los resultados de la operación de algún negocio.

EJB: Enterprise JavaBeans (EJB, por sus siglas en inglés) es una API que forma parte del estándar de construcción de aplicaciones empresariales JEE de la corporación Oracle. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor.

Estados financieros: según las Normas Internacionales de Contabilidad (NIC), constituyen una representación estructurada de la situación financiera y del rendimiento financiero de la entidad. Se entienden por estados financieros el Balance General, el Estado de Ganancias y Pérdidas o Estado de Resultados y otros. En Cuba la información exigida a las empresas que solicitan créditos contiene básicamente el Balance General y el Estado de Resultados del período correspondiente a la fecha de solicitud y al período anterior.

Estado de resultados: es también uno de los estados principales de la Contabilidad, mediante el cual se presenta el volumen total de los ingresos y gastos incurridos por la entidad durante el período que abarca el mismo, con el objetivo de poder conocer si la entidad ha obtenido beneficio o pérdida por la gestión realizada.

JDBC: es un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

JNDI: es una Interfaz de Programación de Aplicaciones (API) de Java para servicios de directorio. Permite a los clientes descubrir y buscar objetos y datos a través de un nombre.

Morosidad: se define como el retraso en el cumplimiento de un pago.

Ratios financieros: también conocidos como indicadores o índices financieros, son razones que permiten analizar los aspectos favorables y desfavorables de la situación económica y financiera de una empresa.

SAGEB: Sistema Automatizado de la Gestión Bancaria.

Scoring: es una de las técnicas más utilizadas en la valoración del riesgo para asignación de límites, basado en la aplicación de técnicas estadísticas de análisis multivariable, con el objetivo de determinar las leyes cuantitativas que rigen la vida económica de la empresa.

ANEXOS

Anexo 1: Representación de la Arquitectura del sistema.

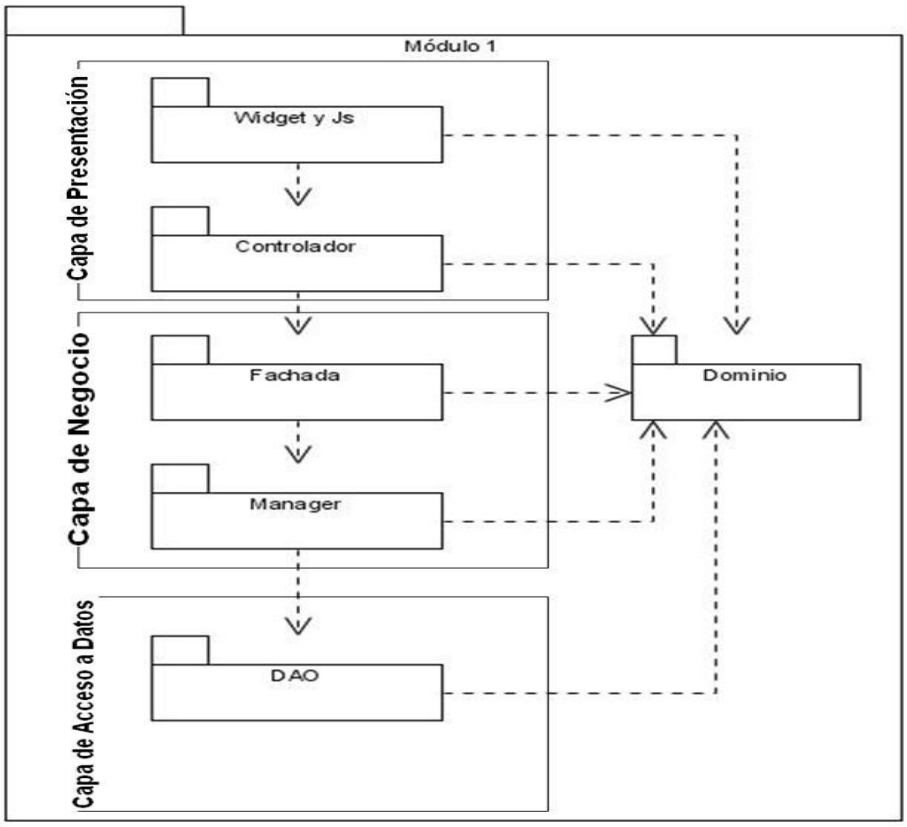


Figura 24: Representación de la arquitectura del sistema.

Anexo 6 Carta de aceptación del BNC.

BNC

14 de Junio de 2012

"Año 54 de la Revolución"

Asunto: Certificación del módulo Análisis de Riesgo del sistema Quarxo desarrollado por la Universidad de las Ciencias Informáticas (UCI) para el Banco Nacional de Cuba (BNC).

A favor de Areanne Leyva Rivera, estudiante perteneciente al proyecto SAGEB, cuyo significado corresponde a: Sistema Automatizado para la Gestión Bancaria.

El módulo Análisis de Riesgo comprueba la correcta integración con el resto de los componentes de la aplicación. Las validaciones de los datos, la aceptabilidad de las operaciones y los niveles de seguridad de la información procesada presentan buen funcionamiento y brindan resultados positivos.

Señalando como elementos positivos, la obtención de nuevas funcionalidades que permitirán agilizar el proceso de análisis de riesgo de crédito en el BNC.

Atendiendo a lo anterior y para que conste ante las instancias pertinentes de la UCI, firmo el documento.

Fraternalmente,

Eduardo Chouza Carrillo

Analista de riesgo