Universidad de las Ciencias Informáticas

Facultad 3

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: Desarrollo del subsistema Análisis de Riesgo del sistema Quarxo.

Autores:

Yordan Anglada Thomas

Yenquiel Hernández Blanco

Tutor:

Ing. Yoan Antonio López Rodríguez

La Habana, julio de 2014

"Año 56 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los _____ días del mes de julio del año 2014.

Yordan Anglada Thomas

_____

Firma del autor

Yenquiel Hernández Blanco

_____

Firma del autor

Yoan Antonio López Rodríguez

_____

Firma del tutor

DATOS DE CONTACTO

Ing. Yoan Antonio López Rodríguez

Ingeniero en Ciencias Informáticas, graduado en julio del 2008

Profesor Asistente

Departamento: Soluciones Financieras y Aduanales

Centro: Informatización de Entidades

Líneas investigativas en las que ha trabajado: Software de gestión y Software educativo

Universidad de las Ciencias Informáticas, Facultad 3

"La programación en bajo nivel es buena para el alma del programador".

John Carmack

Primeramente agradezco mi culminación de estudios a mis padres Noel e Isabel, los cuales me han apoyado y se han sacrificado hasta el último suspiro para que me pudiera graduar, ellos son los principales factores de mi inspiración y consagración, que me han dado la fuerza y el deseo de ser ingeniero, siempre los tengo presente en mis pensamientos y decisiones que voy a tomar en mi vida porque sin su presencia no sería nadie en estos momentos.

Le quiero agradecer a mi abuela Daris que siempre se preocupa por mí tanto en los malos como en los buenos momentos y me cuidó desde que era muy pequeño, a mi tía Amelia que cuando puede me ayuda muchísimo incondicionalmente, a mi tía de crianza María que me ha educado y me ha guiado por el buen camino del estudio y la preparación docente, a mi tutor Yoan que siempre me exigió y me aconsejó en el transcurso del desarrollo de la tesis, a todos mis compañeros de grupo y conocidos que tuve en la universidad.

Finalmente le agradezco a mi novia Dianelys por todo el tiempo que me ha dedicado desde que nos conocimos, todos sus consejos y principalmente todo el amor que me ha brindado.

Yordan Anglada Thomas

Me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo, en especial al Ing. Yoan A. López, tutor de esta investigación, por la orientación, el seguimiento y la supervisión continúa de la misma.

A mis maestros que ayudaron en mi formación profesional, pero sobre todo a la Profesora Hilda, quien ha sido como una segunda madre para mí y a la cual le doy un agradecimiento especial por su invaluable apoyo y confianza.

A mis amigos y a todas las personas que han formado parte de mi vida, porque sin ellos no hubiera sido lo mismo.

A mi familia. Todo mi cariño y mi amor para las personas que hicieron lo imposible en la vida para que yo pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón y mi agradecimiento.

Gracias, eternamente gracias, a todos.

Yenquiel Hernández Blanco

El presente trabajo de diploma se lo dedico a mis padres que siempre me querrán por toda la vida.

Yordan Anglada Thomas

A mis padres, como un testimonio de cariño y eterno agradecimiento por mi existencia, valores morales y formación profesional. Porque sin escatimar esfuerzo alguno, han sacrificado gran parte de su vida para formarme y porque nunca podré pagar todos sus desvelos, ni aún con las riquezas más grandes del mundo. Por lo que soy y por todo el tiempo que les robé pensando en mí…

A todos los que directa e indirectamente ayudaron a la realización de este proyecto.

Yenquiel Hernández Blanco

RESUMEN

Luego de la implantación de la primera versión del sistema Quarxo en el Banco Nacional de Cuba; sistema informático desarrollado en el Centro de Informatización de Entidades de la Universidad de las Ciencias Informáticas, se solicitó por parte del cliente incorporarle al sistema un grupo de procesos que quedaron pendientes en su primera entrega. El proceso que se lleva a cabo para decidir el otorgamiento de un crédito solicitado por un cliente, conocido como proceso de análisis de riesgo de créditos, es uno de ellos. Para su informatización se desarrolló el subsistema Análisis de Riesgo integrado al sistema Quarxo.

El desarrollo del subsistema se realizó utilizando la metodología, los lenguajes y las herramientas definidas en el proyecto Quarxo. Como metodología de desarrollo se siguió el Modelo de Desarrollo de Software v1.1, como ambiente de desarrollo se utilizó la plataforma JEE integrada al Eclipse, como servidor de aplicaciones el Apache Tomcat y como gestor de base de datos SQL Server 2005. Se empleó el framework Spring en la programación de las capas intermedias, conjuntamente con el framework Dojo Toolkit en los efectos visuales de la presentación. Para la persistencia de los datos se utilizó el framework Hibernate.

El desarrollo del subsistema Análisis de Riesgo, permitió ofrecerle al Banco Nacional de Cuba una forma mejorada e integrada a su principal sistema informático de realizar el análisis de riesgo de créditos. Para la validación final del subsistema se realizaron pruebas de caja blanca y pruebas de caja negra, las cuales arrojaron resultados satisfactorios.

PALABRAS CLAVES

Crédito, análisis, riesgo, diseño, implementación.

ÍNDICE

INTRODUCCIÓN

El otorgamiento de créditos realizado en los bancos resulta ser uno de los procesos que más cuidado exige para los mismos. A pesar de que los bancos tienen la misión de inyectar capital en las empresas y con ello contribuir al desarrollo de la economía, en el análisis de riesgo de créditos que se realiza previo al otorgamiento del crédito, es necesario tener total certeza del cumplimiento del reembolso que se acuerde en la firma del contrato, estudiándose para ello de forma exhaustiva la situación financiera de la empresa.

La situación financiera de una empresa está determinada por la evaluación exhaustiva de sus operaciones bancarias (OB) y su información financiera (IF), la cual a su vez está compuesta por un balance general (BG) y un estado de resultados (ER).

El BG muestra contablemente los activos, los pasivos y la diferencia entre estos. El ER consiste en desglosar los gastos e ingresos en distintas categorías, además de mostrarlos antes y después de impuestos.

El Banco Nacional de Cuba (BNC) con el cual la Universidad de las Ciencias Informáticas (UCI) ha venido colaborando desde hace algunos años, es uno de los bancos cubanos que otorgan créditos a empresas.

Actualmente para llevar a cabo el proceso de análisis de riesgo de créditos en el BNC, por cada una de las empresas clientes en capacidad de solicitar créditos, se registra trimestralmente en tablas Excel de forma manual su BG y su ER.

Luego de la implantación del sistema Quarxo desarrollado por la UCI, el cual permite la gestión centralizada de la información para la toma de decisiones; dicho cliente ha solicitado incorporar al sistema la gestión de la información financiera de las empresas de manera que pueda ser usada en conjunto con las operaciones bancarias para el análisis de riesgo de créditos.

Dada la situación problemática anterior surge el siguiente **problema a resolver**: El análisis de riesgo de créditos en el Banco Nacional de Cuba se ve influenciado negativamente por no contar con una gestión centralizada de la información financiera de los clientes.

El **objeto de estudio** de este trabajo se enfocó en el proceso de análisis de riesgo de créditos en los bancos cubanos y el **campo de acción** en el proceso de análisis de riesgo de créditos en el sistema Quarxo.

Para darle cumplimiento a la solución de la problemática planteada se determinó como **objetivo general**: Desarrollar un subsistema que permita realizar el análisis de riesgo de créditos a partir de la gestión centralizada de la información financiera de los clientes en el Banco Nacional de Cuba.

Conformado por los siguientes **objetivos específicos**:

1. Fundamentar la investigación mediante la elaboración del marco teórico para sustentar los conceptos y la propuesta de desarrollo del subsistema Análisis de Riesgo del sistema Quarxo.
2. Desarrollar el subsistema Análisis de Riesgo del sistema Quarxo.
3. Validar la solución presentada.

Con la meta de cumplir con lo planteado anteriormente, se llevaron a cabo las siguientes **tareas**:

1. Investigar sobre los procesos de análisis de riesgo de créditos mediante sistemas informáticos en el mundo.
2. Analizar detalladamente la tecnología y la arquitectura definidas en el proyecto Quarxo.
3. Analizar los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
4. Realizar el modelado del negocio para los procesos de análisis de riesgo de créditos.
5. Capturar los requisitos para el desarrollo del subsistema propuesto.
6. Realizar el diseño del subsistema.
7. Validar el diseño del subsistema.
8. Implementar la solución.
9. Validar la solución presentada.

Resultado esperado: Un subsistema de Análisis de Riesgo para el sistema bancario Quarxo.

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 análisis de riesgo de créditos, así como de varios sistemas informáticos para la gestión bancaria que realizan este proceso. Además, se abordan temas de vital importancia para la realización del subsistema Análisis de Riesgo del sistema Quarxo, señalando lo referente al estudio de la metodología, los lenguajes, las herramientas y los marcos de trabajo, que apoyarán el proceso de desarrollo de software.

Capítulo 2. Modelado del negocio, requisitos, análisis y diseño: Inicialmente se realizan las disciplinas: Modelado del negocio y Requisitos, que permitirán definir las funcionalidades a implementar en el subsistema para responder a las necesidades del cliente; luego se obtiene el modelo de diseño, enfocado en la construcción de diagramas de clases de diseño, diagramas de paquetes y diagramas de secuencias. Posteriormente se lleva a cabo la validación del diseño a partir de las métricas Tamaño Operacional de Clases y Relaciones entre Clases.

Capítulo 3. Implementación y pruebas de la solución propuesta: Se lleva a cabo la implementación del subsistema, donde se explican los estándares de codificación utilizados y las clases principales. Se obtienen además, los artefactos generados en esta etapa del software según la metodología. Finalmente se valida la implementación de los requisitos propuestos a través de pruebas de Caja Blanca y Caja Negra.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se abordan diferentes temas que constituyen la base para efectuar el desarrollo del subsistema Análisis de Riesgo del sistema Quarxo. Se estudian modelos y sistemas informáticos que permiten llevar a cabo el análisis de riesgo de créditos en los bancos del mundo. Además, se analiza la metodología, los lenguajes, las herramientas, los marcos de trabajo y los patrones de diseño definidos para el desarrollo de la solución.

1.2 Sistemas bancarios

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 (Suárez Olano, 2008). La importancia de este sistema es primordial para el desarrollo de la economía, ya que su principal función es suministrar fondos tanto a empresas públicas y privadas, como a personas naturales que los necesitan para que puedan cumplir con los compromisos de pagos contraídos con los proveedores de 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 el funcionamiento del mismo. 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. El proceso de análisis de riesgo de créditos llevado a cabo en los bancos que otorgan créditos a clientes jurídicos es de vital importancia, si se tiene en cuenta la inestabilidad que predomina en las economías a nivel mundial.

1.2.1 Sistema bancario cubano

En la actualidad Cuba se encuentra inmersa en un proceso de transformación económica, con el fin de establecer las bases para fomentar un futuro desarrollo en la economía cubana e

insertarse en los mercados internacionales. Para lograr esta meta se necesita elevar la eficiencia empresarial y perfeccionar la gestión financiera de las empresas (2).

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 (3).

Luego de la implantación del sistema Quarxo desarrollado por la UCI, sistema que personaliza la mayoría de los procesos de esa entidad y permite la gestión centralizada de la información para la toma de decisiones, el BNC ha solicitado incorporar a dicho sistema la gestión de la IF de las empresas de manera que pueda ser usada en conjunto con las OB para el análisis de riesgo de créditos.

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. (Rivas, 2013) 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 (2).

Las OB 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 (4). 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 crediticios 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 permitirán tener una mejor visión sobre el cliente y la capacidad para poder pagar dicho crédito (5).

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.

1.2.4 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 (6).

Consiste en la probabilidad de que ocurra un siniestro financiero por incapacidad de pago de los deudores del banco, tanto en forma individual como en forma consolidada (7).

Es la posibilidad de que una entidad incurra en pérdidas y disminuya el valor de sus activos, como consecuencia de que sus deudores o contrapartes 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 créditos 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 plataformas para un sano crecimiento de la cartera crediticia (1).

1.3 Modelos para el análisis de riesgo de créditos

1.3.1 Modelos tradicionales

A partir de los años 60 transcurrieron varias décadas en las que el análisis de riesgo de créditos 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, quien 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 al cliente. En ocasiones estos elementos traen consigo consecuencias negativas.

Los elementos analizados por este modelo son los siguientes:

1. **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.
2. **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.

3. **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 pueden 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.
4. **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.
5. **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 (8).

El enfoque tradicional depende en gran medida de la información financiera presentada por la empresa solicitante del crédito y se basa en la experiencia de los analistas de riesgos.

1.3.2 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:

- **Modelo Z-Score**

En este modelo 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 de créditos que tienen muy en cuenta el riesgo de mercado (27).

- **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 (7). 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 ayuda a predecir la probabilidad de pago con base en la cuantificación de las características incluidas en la base de datos.

1.4 Análisis de riesgo de créditos en Cuba

En Cuba los bancos que conceden créditos a empresas realizan el análisis de riesgo de créditos por medio del modelo tradicional. 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 (8).

Se plantea que un analista de riesgos desprovisto de herramientas informatizadas puede ofrecer diferentes respuestas ante situaciones similares. El subsistema a desarrollar constituye una herramienta 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 informáticas 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 mismo, se pretende continuar utilizando el enfoque tradicional, apoyando al analista de riesgo con herramientas informáticas.

1.5 Sistemas informáticos para el análisis de riesgo de créditos

En las últimas décadas se han venido desarrollando una serie de sistemas informáticos que permiten la gestión de los riesgos de crédito en las entidades. Para su estudio en la presente investigación se han tenido en cuenta una serie de indicadores. Entre ellos se tienen: la nacionalidad del software, si es libre o propietario y el enfoque de análisis de riesgo de créditos que presenta. A partir de esos indicadores y persiguiendo extraer información valiosa para el desarrollo del subsistema de la presente investigación, se hace un estudio de diferentes sistemas internacionales y nacionales.

1.5.1 Sistemas informáticos internacionales

En el caso de los sistemas internacionales es importante mencionar que los sistemas líderes para el análisis de riesgo de créditos, donde se incluyen los estudiados, son estadounidenses y propietarios, con lo que eso significa para Cuba. A continuación se presentan los sistemas internacionales:

- **@Risk**

Entre sus principales características se tienen:

1. Permite la gestión de gran número de riesgos, en cualquier tipo de entidad.
2. Usa el enfoque de los modelos modernos.
3. Basa el análisis de riesgo de créditos en la simulación, se basa en modelación de ambientes.
4. Es software propietario.
5. No es multiplataforma.

- **Power Risk**

Es un software de riesgos financieros creado por la empresa Scalar Consulting. Entre los módulos que desarrolló se encuentra el "Power Risk–Scoring". Entre sus principales características se tienen:

1. Generar puntajes para operaciones prospectivas por tipo de cartera.
2. Facilidad de generar variables cualitativas y numéricas por cada tipo de cartera.
3. Proveer probabilidades de incumplimiento ex-ante.
4. Usa el enfoque de los modelos modernos.
5. Incluye cálculo de margen de error.
6. Aplicable tanto a personas como a empresas.
7. Es software propietario y el pago de sus licencias es costoso.
8. Utiliza la experiencia de la institución ante los nuevos créditos que se presentan (10).

De manera general los sistemas expuestos constituyen buenas soluciones informáticas para el mundo financiero por la gama de funcionalidades que brindan. Sin embargo, los altos costos de las licencias, el hecho de ser sistemas enfocados en modelos modernos y estadounidenses no son características coherentes a los objetivos perseguidos en la presente investigación. Por ello se decidió hacer hincapié en el estudio de sistemas nacionales que cumplían con los indicadores deseados.

1.5.2 Sistemas informáticos nacionales

Se decidió que el estudio de los sistemas informáticos nacionales debía estar intencionado hacia el análisis de las herramientas de desarrollo que se emplearon y las funcionalidades que brindan. A continuación se hace referencia al estudio realizado sobre el sistema del Banco Internacional de Comercio SA (BICSA).

Sistema del BICSA

El sistema para el análisis de riesgo de créditos del BICSA desarrollado en .Net por informáticos de Desoft Pinar del Río, sigue el enfoque tradicional y tiene como entradas además del balance general y del estado de resultados, el flujo de caja de la empresa. El flujo de caja le permite al analista de riesgo tener una visión de la empresa en caso de que finalmente se le otorgara el crédito y se encontrara pagando las cuotas acordadas.

El sistema del BICSA utiliza para el análisis de riesgo de créditos el historial que mantiene del comportamiento de los clientes en los pagos de créditos anteriores, constituyendo esto el Carácter de la empresa que es uno de los 5 elementos que caracterizan al enfoque tradicional. Asimismo, el sistema permite utilizar en el análisis de riesgo de créditos el comportamiento actual de la empresa dado por sus operaciones bancarias en el propio banco. Otro aspecto significativo que tiene el sistema del BICSA es que los rangos de los indicadores que se calculan para determinar la situación financiera del cliente no son fijos, sino que difieren según el tipo de cliente.

De los sistemas estudiados, el sistema del BICSA fue el que más aportó a la solución del presente trabajo, en cuanto a la definición de funcionalidades para el subsistema Análisis de Riesgo. Del mismo se tomaron aspectos tales como: el trabajo con el flujo de caja; la creación del historial de créditos y la comprobación del resto de las operaciones que la empresa mantiene con el banco. Es válido señalar que en el caso del BNC, a diferencia del BICSA, las empresas no mantienen la mayor parte de sus operaciones en ese banco, por lo que resulta muy importante que se recoja la valoración de las empresas en las reuniones que se realizan entre los analistas de riesgos de todos los bancos de la capital. También se tomó del sistema del BICSA la idea de clasificar los rangos de los indicadores para los clientes según la institución a la cual pertenecen.

Por otra parte, en cuanto a las herramientas de desarrollo, solo se conoció que el sistema del BICSA fue desarrollado en la plataforma de desarrollo .Net y que se usó como gestor de base de datos SQLServer 2005, ya que se dificultaron los intercambios con los desarrolladores de Desoft Pinar del Río. No obstante, se acordó reforzar los encuentros de intercambio en lo adelante para mejorar las soluciones informáticas.

1.6 Metodología de desarrollo de software

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 (11).

Para el desarrollo del presente trabajo se usará como metodología de desarrollo el Modelo de Desarrollo de Software v1.1, modelo establecido en el proyecto Quarxo del cual es parte el

subsistema del presente trabajo. El mismo será utilizado para el desarrollo de la solución, puesto que incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del centro y define cada uno de los artefactos a generar en cada momento, independientemente de las herramientas o métodos que se utilicen para ello.

1.6.1 Modelo de Desarrollo de Software v1.1

La metodología del centro CEIGE plantea el desarrollo de dos fases: Inicio y Desarrollo. A continuación se da una breve panorámica de ambas fases, haciendo especial énfasis en la segunda que es la que guía el trabajo de la presente investigación.

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

Desarrollo: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

El objetivo de esta fase es:

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

Como hito de la fase se tiene:

- Producto liberado por entidad certificadora de calidad.

Según se establece en el modelo durante la fase Desarrollo se ejecutan las disciplinas: Modelado del negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación.

Las principales características del Modelo de Desarrollo de Software v1.1 son:

1. **Centrado en la arquitectura**: la arquitectura determina la línea base y los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las

prioridades en la producción y resuelve las necesidades tecnológicas y de soporte para el desarrollo.

2. **Orientado a componentes**: las iteraciones son orientadas según la significación arquitectónica de los componentes. Los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan. El componente es la unidad de medición y ordenamiento de las iteraciones.

3. **Iterativo e incremental**: las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la iteración, permitiendo de esta manera la evolución incremental del producto.

Ciclo de vida de los proyectos del CEIGE según el modelo

El ciclo de vida de los proyectos del CEIGE tiene en cuenta las actividades de cada una de las fases y áreas de procesos que plantea el nivel dos de CMMI establecido en la UCI. Abarca el total de acciones que se realizan en las distintas líneas de desarrollo para la elaboración del servicio o producto final, sin embargo, se debe adaptar a las características particulares del proyecto, ya que puede ocurrir que no resulte necesario la ejecución de determinada disciplina, o la elaboración de determinados artefactos. A continuación se representa el ciclo de vida de los proyectos (*ver Figura 1.1*) según el modelo de desarrollo del centro CEIGE.

Figura 1.1. *Modelo de desarrollo del centro CEIGE.*

Explicación de las disciplinas de la fase Desarrollo

1. **Modelado del negocio**: Es la disciplina destinada a comprender los procesos de negocio de la organización. Se comprende la manera en que funciona el negocio que se desea automatizar para tener garantía de que el software desarrollado va a cumplir su propósito.
2. **Requisitos**: El esfuerzo principal en la disciplina de requisitos es desarrollar un modelo del sistema que se va a construir. Incluye un conjunto de artefactos que describen todas las interacciones que tendrán los usuarios con el software y que responden a los requisitos funcionales del sistema. Se especifican los requisitos funcionales y no funcionales.
3. **Análisis y diseño**: Durante esta disciplina es modelado el sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima disciplina. Los artefactos generados en esta etapa son más formales y específicos de una implementación. En caso de llevarse a cabo la reutilización de componentes de software ya desarrollados, durante esta disciplina se ajusta el modelado existente a los requisitos actuales.
4. **Implementación**: A partir de los resultados del análisis y diseño se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ejecutables y similares. Al reutilizar componentes de software ya implementados se lleva a cabo el desarrollo necesario para ajustar a los requisitos actuales y posteriormente realizar la integración de los componentes.
5. **Pruebas internas**: Durante esta disciplina se desarrollan las pruebas del grupo de calidad del centro verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software, es decir, requisitos que el producto debería cumplir y que aún no los cumple.
6. **Pruebas de liberación**: Se aplican pruebas diseñadas e implementadas por el Laboratorio Industrial de Pruebas de Software a todos los entregables de los proyectos antes de ser proporcionados al cliente para su aceptación (37). En la presente investigación la validación llegará solo hasta las pruebas internas en las cuales personal del centro CEIGE validará las funcionalidades implementadas en el subsistema.

De los artefactos que propone el Modelo de Desarrollo de Software v1.1 en sus disciplinas durante la fase Desarrollo, se realizarán en la presente investigación los siguientes:

1. Modelado del negocio:

- Descripción de procesos de negocio.

- Modelo conceptual.
- Reglas del negocio.

2. Requisitos:

- Especificación de requisitos de software.
- Documento de salida del sistema.

3. Análisis y diseño:

- Modelo de datos.
- Modelo del diseño, que abarca los diagramas de clases, de paquetes y de interacción.
- Documento de casos de pruebas.

4. Implementación:

- Modelo de componentes.

5. Pruebas internas:

- Pruebas de unidad, Método Caja Blanca, Técnica: Camino Básico a través del JUnit.
- Pruebas de integración, Método Caja Negra, Técnica: Partición de equivalencia, A través de los casos de pruebas.

1.7 Lenguajes

Un lenguaje informático es el idioma utilizado por los ordenadores con el objetivo de transmitir información mediante equipos de cómputo. Estos pueden ser clasificados en varios tipos, tales como: lenguajes de programación, modelado, consulta, sonido, gráfico, marcas y pseudocódigos. A continuación se describirán los diferentes lenguajes de modelado y de programación utilizados en el desarrollo del subsistema Análisis de Riesgo, los cuales fueron definidos por el proyecto Quarxo.

1.7.1 Lenguajes de modelado y notación

Se denomina lenguaje de modelado de objetos al conjunto estandarizado de símbolos y las distintas combinaciones de ellos para modelar un diseño de software (14).

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 (15). Básicamente facilita a los

desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados.

BPMN (Business Process Managment Notation): es una notación gráfica que describe la lógica de los pasos de un proceso de negocio. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma BPMN define la notación y semántica de un Diagrama de Procesos de Negocio (BPD, en inglés: Business Process Diagram). El modelado en BPMN se realiza mediante diagramas muy simples con un conjunto muy pequeño de elementos gráficos. Con esto se busca que para los usuarios del negocio y los desarrolladores técnicos sea fácil entender el flujo y el proceso (42).

Para la realización de los diagramas de procesos de negocios se utilizó la notación BPMN, y para el resto de los artefactos que se realizaron según se establece en el modelo de desarrollo seguido, se utilizó el lenguaje UML.

1.7.2 Lenguajes en el lado del servidor

Java 6: 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 las sentencias de bajo nivel y el 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 (16).

1.7.3 Lenguajes en el lado del cliente

JSP[1] es un lenguaje para la creación de sitios web dinámicos, orientado a desarrollar páginas web en Java y es multiplataforma. Desarrollado por Sun Microsystems para programar aplicaciones web potentes. Posee un motor de páginas basado en los servlets de Java y necesita tener instalado un servidor Tomcat.

JavaScript: es un lenguaje interpretado, es decir, que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con XHTML. No es

[1] *Java Server Page*

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 rápido desarrollo y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox (19).

Se definió en el proyecto el uso de Javascript para la validación de los datos de entrada y JSP por su integración al lenguaje base Java, para la creación de las páginas web.

1.8 Herramientas

Las herramientas son todos aquellos programas utilizados en la creación de un determinado producto o sistema informático. Es de vital importancia, luego de haber definido la metodología que guiará el desarrollo del producto y los lenguajes a utilizar en el diseño e implementación del sistema, definir las herramientas adecuadas para la realización del software.

1.8.1 Herramienta de modelado CASE

Las herramientas de modelado 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 documentación a partir de un diseño dado. En sentido general estas herramientas intentan dar ayuda automatizada y sirven de apoyo a la metodología de desarrollo empleada (24). A continuación se brinda una explicación de las herramientas que se usarán, las cuales fueron definidas por el proyecto Quarxo.

Visual Paradigm 8.0: herramienta multiplataforma para el modelado UML que soporta el ciclo de vida completo del desarrollo de software. 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 (25).

1.8.2 Entorno integrado de desarrollo

Eclipse 3.5 GALILEO: es una herramienta de código abierto y multiplataforma desarrollado por la compañía IBM (International Business Machines). Emplea plugins 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 plugins 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 (27).

Plataforma JEE (Java Enterprise Edition): 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 y varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras (17).

Después de un análisis se decidió por la dirección del proyecto en un inicio el uso de este IDE sobre la plataforma JEE por las características que presenta y la experiencia de los desarrolladores con los mismos.

1.8.3 Contenedor web

A continuación se describe brevemente el Apache Tomcat 6.0.35 que se empleará en la presente investigación por ser el contenedor web definido en el proyecto Quarxo y tener una serie de bondades necesarias para el desarrollo satisfactorio del subsistema.

Apache Tomcat 6.0.35: es un contenedor de Servlets bajo la filosofía del código abierto licenciado con Apache Software 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 (28).

1.8.4 Gestor de base de datos

Microsoft SQL Server 2005: es un servidor de base de datos basado 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 (Standard Query Language) y el T-SQL (Transact-SQL) (29). Requiere para su funcionamiento un sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización.

Este gestor de base datos aunque es un software privativo, se seleccionó para el trabajo tomando en cuenta la petición del cliente, por la familiarización que presenta con la herramienta y considerando los procedimientos que ya estaban almacenados en este y que por cuestiones de tiempo no podrían ser implementados en otro gestor.

1.9 Marco de trabajo

Un marco de trabajo 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 (20).

1.9.1 Marcos de trabajo en el lado del servidor

Spring Framework 2.5.6: 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 la inversión de control e inyección de dependencia. El framework está dividido en módulos para su correcto funcionamiento, tal como se muestra a continuación:

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

- **Spring Context**: es el que consagra a Spring como un framework ya que ofrece soporte para la internacionalización y la aplicación de eventos de ciclo de vida. Brinda soporte a servicios como correo electrónico, acceso a Java Naming and Directory Interface (JNDI) e integración con Enterprise JavaBeans (EJB) (21).
- **Spring DAO**: provee un trabajo con JDBC (Java Database Connectivity) 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 ORM**: brinda el soporte necesario para la integración con el framework Hibernate.
- **Spring AOP**: ofrece un extenso soporte para la Programación Orientada a Aspectos, permitiendo definir además la política transaccional y de seguridad de una aplicación.
- **Spring Web**: es el encargado de crear el contexto para aplicaciones web. Incluye soporte para una variedad de tareas tales como: la subida de archivos, la vinculación de parámetros de las peticiones a objetos del negocio, entre otras.
- **Spring Web MVC**: ofrece gran soporte para el desarrollo de aplicaciones web utilizando la filosofía Modelo-Vista-Controlador (MVC). Emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio.

El marco de trabajo Spring fue seleccionado por el equipo de arquitectura del proyecto Quarxo debido a que posee una comunidad de gran prestigio que facilita la retroalimentación, se acopla perfectamente a la plataforma JEE y su uso propicia la aplicación del patrón de arquitectura MVC.

1.9.2 Marcos de trabajo en el lado del cliente

Dojo Toolkit 1.3: 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 (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 (Document Object Model). Ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS (Cascading Style Sheet) y JavaScript para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: Dojo, Dijit, Dojox y Dijitx (23).

Dojo fue seleccionado por el equipo de arquitectura debido a su fácil integración con Spring MVC y teniendo en cuenta que el proyecto SIGEP desarrolló una gran cantidad de componentes y funciones, las cuales se podían reutilizar en el desarrollo del sistema Quarxo.

1.9.3 Marcos de trabajo para el acceso a datos

Hibernate 3.5: es una solución ORM (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. Permite la ejecución de consultas SQL (22).

Hibernate fue seleccionado principalmente por la sólida integración con Spring Framework, la facilidad que brinda en el control sobre las sesiones y el gran número de métodos para el trabajo con los datos que incluye la clase HibernateTemplate.

1.10 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 (15). Existen varios grupos principales en los que estos se pueden agrupar.

1.10.1 Patrones GRASP

Los Patrones Generales para Asignar Responsabilidades (en inglés: GRASP), 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 y Controlador, los cuales se explican a continuación:

1. **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.
2. **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.
3. **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.
4. **Bajo acoplamiento**: es un patrón que estimula asignar una responsabilidad de modo que no se produzcan los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también más reutilizables, que acrecientan la oportunidad de una mayor productividad.
5. **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 (15).

1.10.2 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:

1. **Fachada**: patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase. 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 manera 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.
2. **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.

1.10.3 Patrones de acceso a datos

DAO (Data Access Object): tiene como objetivo fundamental abstraer y encapsular todos los accesos a la fuente de datos aislando los objetos de negocio de una implementación particular de la persistencia. La ventaja de usar objetos de acceso a datos está dada en que cualquier objeto de negocio no requiere conocimiento directo del destino final de la información que manipula y que se puede cambiar fácilmente de fuente de datos. Este patrón será utilizado en la creación de clases de acceso a datos con la función de servir como intermediarias entre las clases del negocio y la fuente de datos.

1.11 Conclusiones parciales

Luego del desarrollo del capítulo 1 se concluye lo siguiente:

- Del estudio realizado a sistemas que se utilizan a nivel internacional para informatizar el proceso de análisis de riesgo de créditos, se constató que a pesar de las buenas prestaciones que brindan para el mundo financiero, los altos costos de sus licencias y el hecho de enfocarse en modelos de análisis de riesgo de créditos modernos, no los hacen tan atractivos para la presente investigación, de ahí que la misma se centrara en el estudio de sistemas nacionales.
- El sistema del BICSA fue el que más aportó a la solución del presente trabajo. Del mismo se tomaron numerosos aspectos que fueron tenidos en cuenta para el desarrollo del subsistema Análisis de Riesgo del sistema Quarxo.
- El estudio de las metodologías y herramientas definidas en el proyecto Quarxo para el desarrollo del subsistema Análisis de Riesgo, confirmó la validez del uso de las mismas para lograr los objetivos propuestos.

CAPÍTULO 2: MODELADO DEL NEGOCIO, REQUISITOS, ANÁLISIS Y DISEÑO

2.1 Introducción

La principal aspiración del capítulo es transformar los requisitos funcionales en el diseño de la solución propuesta, permitiendo de esta forma 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 disciplina Implementación, 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 las métricas Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC).

2.2 Modelado del negocio

Es la fase del modelo de desarrollo de software orientada a comprender la dinámica y la estructura de la organización. En ella se describe detalladamente cómo se llevan a cabo los procesos del negocio que se desean automatizar y de esta forma garantizar que el software desarrollado cumple con el propósito propuesto. Su principal objetivo es asegurarse de que los clientes, usuarios y desarrolladores poseen un total entendimiento de la organización.

Primeramente se debe desarrollar el modelo de procesos de negocio y a partir de este derivar los posibles requisitos del sistema. Con esto se logra que el producto desarrollado responda a las condiciones y las necesidades reales del cliente.

2.2.1 Modelo de procesos de negocio

Un proceso de negocio representa una función que transforma los flujos de datos de entrada en uno, o varios flujos de datos de salida. Cada proceso de negocio debe ser capaz de generar los datos de salida a partir de los de entrada. Cada entrada representa un requisito que debe ser identificado antes de aplicarle cualquier función. Estos procesos describen cómo se lleva a cabo el trabajo realizado en la organización y se caracterizan por ser: observables, medibles mejorables y repetitivos.

El modelado de procesos de negocio es usado para entender mejor el funcionamiento de una organización, documentar y publicar los procesos buscando una estandarización en la organización, para lograr eficiencia en las operaciones e integrar soluciones a la arquitectura.

La técnica utilizada para realizar el estudio de los procesos de negocio de la institución e identificar las necesidades reales de los clientes en el BNC fue la entrevista. La misma consiste en recopilar toda la información necesaria mediante el uso de preguntas a los analistas de riesgo de crédito de forma verbal. Es una forma de dialogar y llegar a un consenso sobre cómo se llevan a cabo los respectivos procesos en la entidad. A continuación se presentan los procesos de negocio definidos en el BNC para el análisis de riesgo de crédito:

1. Entrar información financiera.
2. Calcular indicadores contables.
3. Elaborar dictamen de riesgo.

A partir de los procesos identificados se desarrolló el modelado del negocio y para ello se generaron una serie de artefactos que condujeron a la captura de los requisitos del cliente: descripción de los procesos de negocio, modelo conceptual y reglas del negocio. Dichos artefactos fueron validados posteriormente por los funcionales entrevistados.

De los procesos antes mencionados, fue seleccionado Entrar información financiera para exponer de manera sencilla cada disciplina de la fase Desarrollo del Modelo de Desarrollo de Software v1.1. Los artefactos desarrollados para los restantes procesos podrán ser consultados en el repositorio del proyecto Quarxo. A continuación serán mostrados los artefactos generados para el proceso estudiado durante la disciplina Modelado del negocio.

2.2.1.1 Descripción del proceso de negocio

La descripción del proceso permite describir las reglas a tener en cuenta a la hora de transformar las entradas en salidas. Indican el proceso a realizar y la transformación de los datos, no el algoritmo, el cual se selecciona en la disciplina de Análisis y diseño. Para apoyar el estudio se consultaron a los especialistas en el tema correspondiente. A continuación se presenta la descripción del proceso de negocio Entrar información financiera (*ver Tabla 2.1*).

Objetivo	Obtener la información financiera del cliente para poder realizar posteriormente su análisis.
Evento(s) que lo genera(n)	Solicitud de información financiera.
Pre condiciones	La información financiera es traída por el cliente.
Marco legal	N/A
Clientes internos	N/A
Clientes externos	N/A
Entradas	Información financiera del cliente.
Flujo de eventos	
Flujo básico Entrar Información financiera	
1. Entregar información financiera:	El cliente entrega la información financiera.
2. Recibir información financiera:	El usuario recibe la información financiera traída por el cliente.
3. Entrar información financiera:	El usuario introduce la información financiera manualmente en un libro Excel.
Pos-condiciones	
1.	La Información financiera queda registrada.
Salidas	
N/A	

Tabla 2.1. *Descripción del proceso de negocio Entrar información financiera.*

2.2.1.2 Diagrama del proceso de negocio

Los diagramas de procesos son representaciones graficas que se obtienen para cualquier tipo de proceso. Básicamente describen paso a paso toda una secuencia de actividades que se siguen dentro de un procedimiento.

Actualmente en el BNC es posible registrar la información financiera de un cliente jurídico, pero resulta un proceso engorroso para el analista de riesgo de crédito, ya que debe realizar todo el proceso de forma manual. El siguiente diagrama de proceso (*ver Figura 2.1*) muestra lo anteriormente descrito.

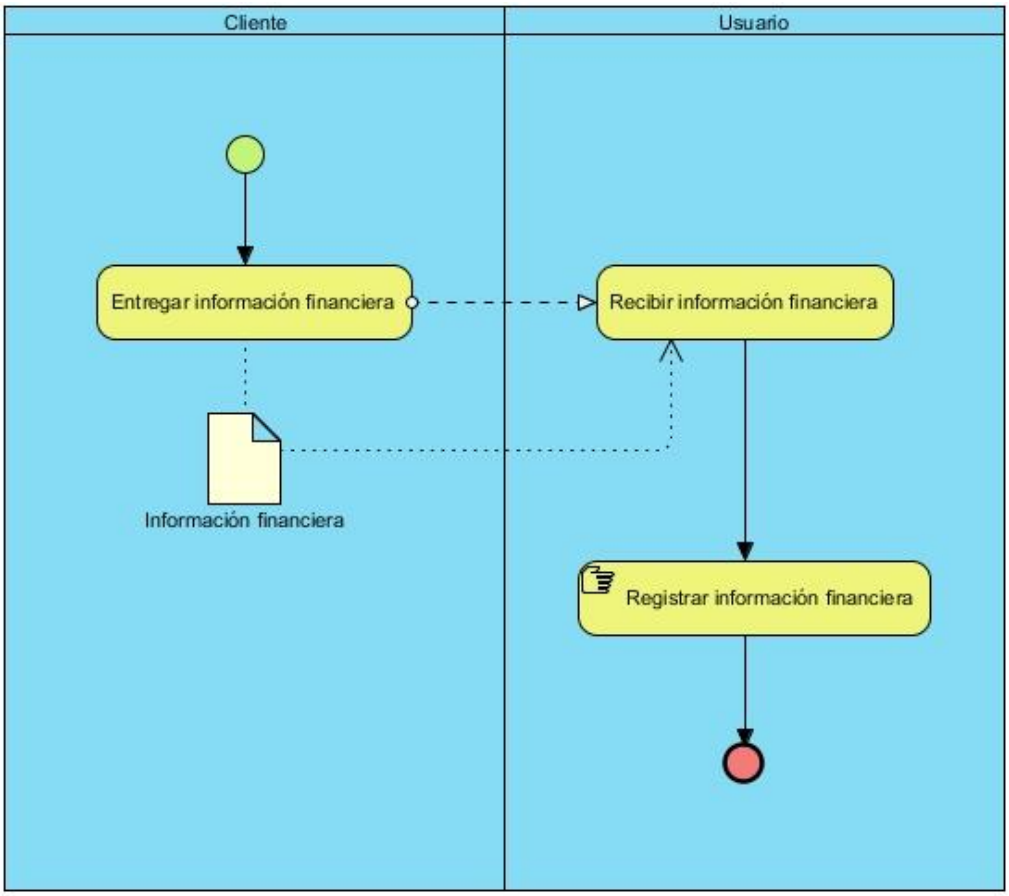

Figura 2.1. *Diagrama del proceso de negocio Entrar información financiera.*

2.2.2 Modelo conceptual

Un modelo conceptual es una representación visual de las clases conceptuales del dominio. Explica los conceptos significativos indicando las características del proceso en la investigación y cómo se relacionan en la descripción del problema. Su principal objetivo es establecer un vocabulario común para lograr una mejor comprensión de las clases principales y facilitar la posterior captura de los requisitos.

Para la elaboración del modelo conceptual correspondiente al proceso Entrar información financiera (*ver Figura 2.2*) se identificaron las clases conceptuales del negocio, sus atributos y las relaciones existentes entre dichas clases. A continuación se muestra el modelo conceptual con los conceptos identificados.

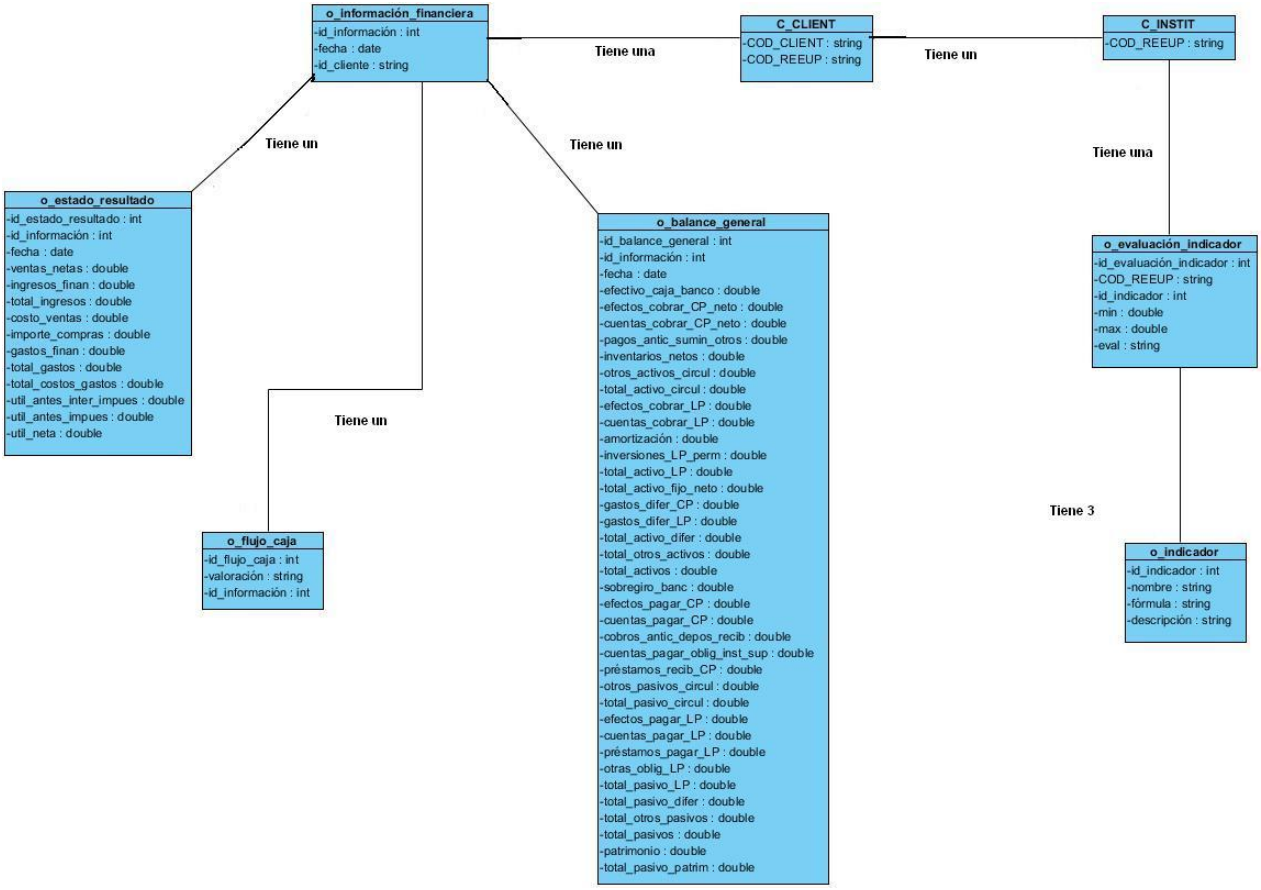

Figura 2.2. *Modelo conceptual del proceso Entrar información financiera.*

2.2.3 Reglas del negocio

Las reglas del negocio son restricciones de vital importancia para el modelo del negocio, ya que proporcionan el soporte para la dirección de las actividades que se aplican a lo largo de los procedimientos. Definen y delimitan cada aspecto del negocio con el objetivo de establecer una estructura. Estas deben ser lo más explicitas posibles y escritas en un lenguaje natural para su total comprensión.

Existen diferentes tipos de reglas dentro de las que se encuentran:

- Reglas textuales.
- Reglas del modelo de datos.
- Reglas de relación.
- Reglas de derivación.

Las reglas del negocio se podrán consultar en el repositorio del proyecto Quarxo.

2.3 Requisitos

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.

2.3.1 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 las reglas del negocio o de la propia interacción de los usuarios (31).

Inicialmente en encuentros con el cliente se realizó una especificación de requisitos tal como se muestra a continuación:

1. RF1: Registrar información financiera
2. RF2: Actualizar información financiera
3. RF3: Consultar información financiera
4. RF4: Eliminar información financiera
5. RF5: Buscar información financiera
6. RF6: Registrar flujo de caja
7. RF7: Actualizar flujo de caja
8. RF8: Consultar flujo de caja
9. RF9: Eliminar flujo de caja
10. RF10: Buscar flujo de caja
11. RF11: Registrar indicador
12. RF12: Actualizar indicador
13. RF13: Consultar indicador
14. RF14: Eliminar indicador
15. RF15: Buscar indicador
16. RF16: Calcular indicadores

Sin embargo a medida que se fue desarrollando el presente trabajo la especificación de requisitos fue sufriendo cambios tales como:

- Debido a que era necesario realizar una tipificación de los rangos de evaluación de los indicadores según el tipo de empresa, lo cual requería de mayor tiempo, se decidió que los indicadores fueran configurados directamente en la base de datos y dejar para futuras versiones la actualización dinámica a través del sistema de los indicadores.
- Con el estudio del sistema del BICSA se decidió introducir solamente una valoración del flujo de caja en conjunto con el balance general y el estado de resultados, puesto que en el país actualmente cada empresa difiere en su flujo de caja y no es posible crear una forma estandarizada para registrarlo. En la valoración del flujo de caja el analista de riesgo puede registrar en el sistema: si se encuentra o no plasmado en el flujo de caja entregado por la empresa, el crédito solicitado; así como si el flujo de caja proyectado aparece negativo.

A partir de lo expuesto anteriormente se proponen para el presente trabajo los requisitos siguientes:

1. RF1: Registrar información financiera (ver Anexo #1)
2. RF2: Actualizar información financiera (ver Anexo #2)
3. RF3: Consultar información financiera (ver Anexo #3)
4. RF4: Eliminar información financiera (ver Anexo #4)
5. RF5: Buscar información financiera
6. RF6: Calcular indicadores

De los requisitos anteriores se decidió seleccionar Registrar información financiera, en correspondencia al proceso de negocio Entrar información financiera, para exponer cada uno de los artefactos generados en esta disciplina. Los restantes podrán ser consultados en el repositorio del proyecto Quarxo.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales (RNF) son restricciones de los servicios o funciones ofrecidas por el sistema. Especifican criterios que pueden usarse para calificar las funcionalidades en lugar de sus comportamientos específicos. Definen propiedades del sistema, del proyecto o del servicio del soporte, que no son requeridas junto con la especificación del sistema, sino como una restricción del software.

De acuerdo con las características sobre las cuales se desarrolla este subsistema, se toman en cuenta los requisitos no funcionales correspondientes al sistema Quarxo. Sus descripciones pueden consultarse en el repositorio del mismo.

2.3.3 Descripción de requisitos

Para ver la descripción del requisito funcional Registrar información financiera, ver Anexo #1.

2.3.4 Validación de los requisitos

La validación de los requisitos se llevó a cabo con el objetivo de garantizar que los mismos fueran descritos correctamente y cumplieran con las necesidades del cliente. Para ello se utilizaron las siguientes técnicas:

- **Revisión técnica por el equipo de analistas principales**: se verificó la construcción correcta de los artefactos correspondientes en la ingeniería de requisitos.
- **Revisión funcional por los especialistas del BNC**: se le presentaron a los especialistas funcionales del BNC los prototipos elaborados durante la especificación de requisitos.
- **Revisión técnica de artefactos por el equipo de calidad**: se realizaron revisiones a los artefactos por parte del equipo de calidad del proyecto y se corrigieron las no conformidades identificadas hasta la liberación de la documentación.

2.4 Análisis y diseño

De forma general en el análisis se modela el sistema, lo que contribuye a la obtención de una arquitectura estable y sólida 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 cabo por diferentes equipos de desarrollo (2).

2.4.1 Modelo de datos

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general permite describir la estructura, así como los datos, sus relaciones, las restricciones de integridad y las operaciones de manipulación (33).

Teniendo en cuenta los requisitos funcionales del módulo Información financiera, se hizo necesario crear un modelo para la representación de los datos persistentes que son manejados.

A continuación se puede observar el modelo de datos (*ver Figura 2.3*) que se realizó para representar los datos que son manipulados en el proceso de análisis de riesgo de créditos.

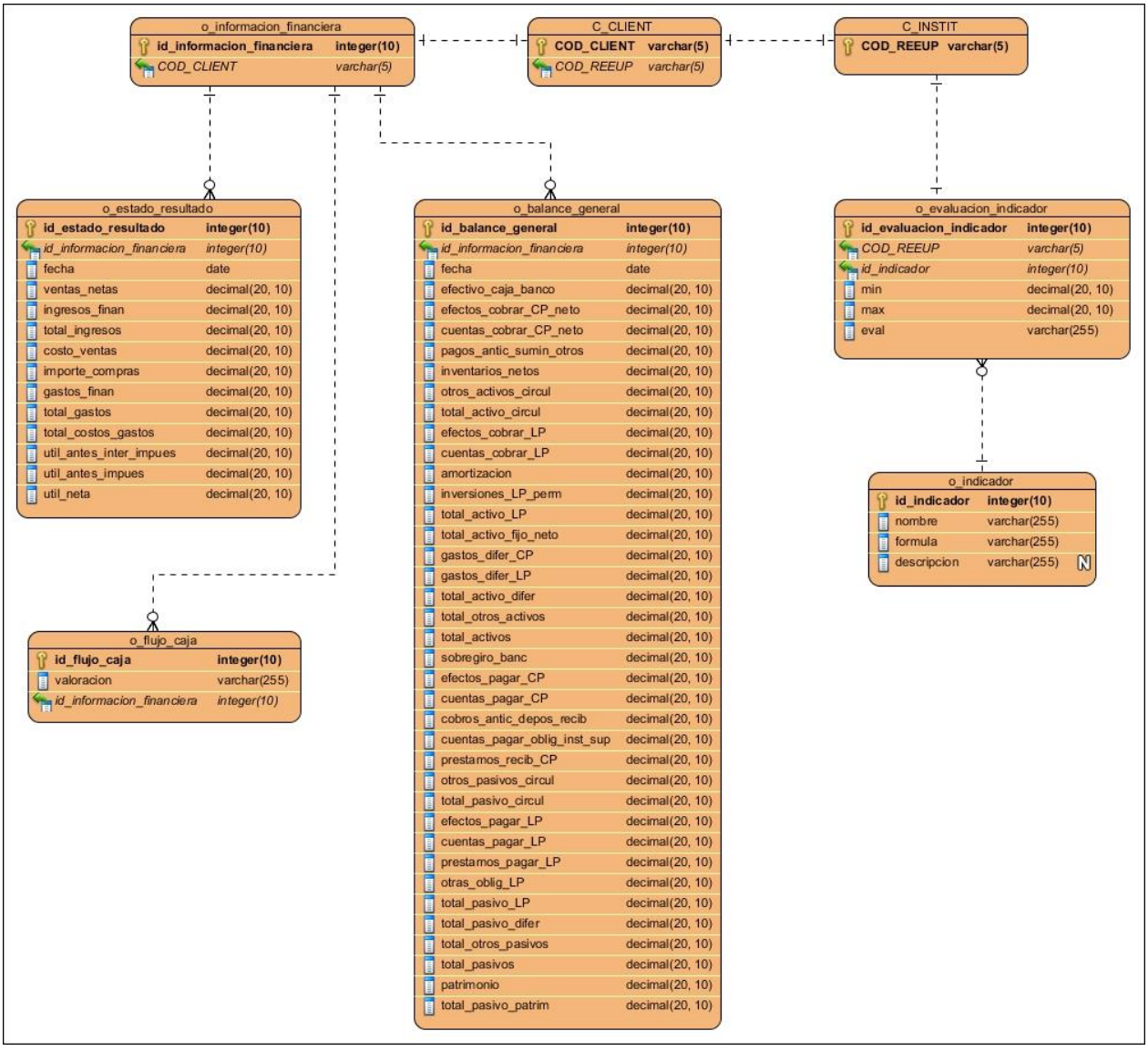

Figura 2.3. *Modelo de datos del subsistema Análisis de Riesgo.*

2.4.2 Arquitectura del sistema

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 definida para el desarrollo del sistema Quarxo (*ver Figura 2.4*) está basada en una arquitectura de tres capas lógicas fundamentales: capa de presentación, capa de negocio, capa de acceso a datos y una capa transversal a las otras con los objetos del dominio (12).

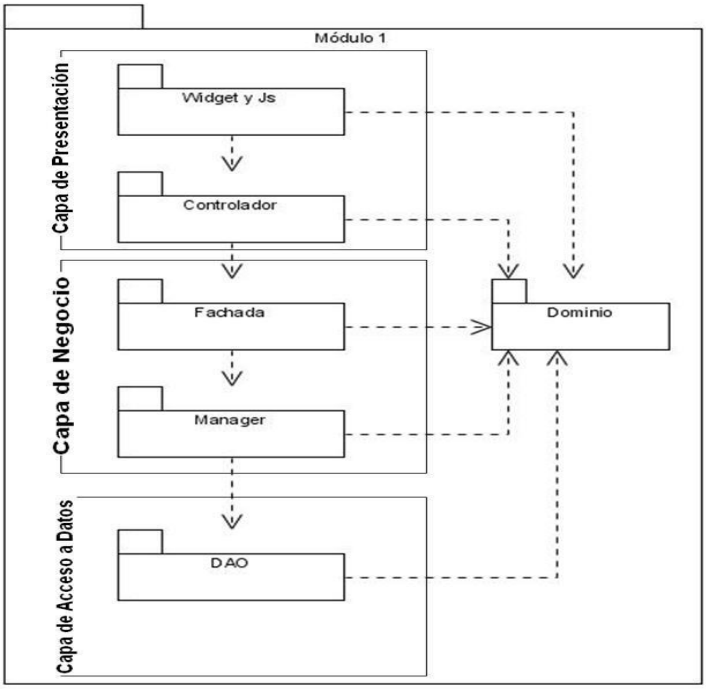

Figura 2.4. *Estructura de las capas del sistema.*

2.4.2.1 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. La capa de presentación está relacionada con la capa de negocio y la capa de dominio.

2.4.2.2 Capa de negocio

Está divida en dos subcapas: Facade y Manager. La Facade es el punto de intercambio entre la capa de presentación y la capa de negocio. Esta capa no tendrá lógica de negocio, sino que agrupará 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.

2.4.2.3 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, permitiendo así la persistencia y el acceso a la información. La interacción con la capa de negocio se realiza a través de interfaces.

Se utilizará el framework de persistencia Hibernate y los módulos 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 la integración con Hibernate y la utilización del patrón DAO.

Cuando la interacción con la base de datos se realiza a través de los procedimientos almacenados se utilizará Spring JDBC. El mismo se utilizará mediante el componente desarrollado por el proyecto para facilitar las invocaciones a funciones y procedimientos almacenados. Este componente permite que de una misma clase DAO se pueda acceder a varias funciones y procedimientos de forma transparente al desarrollador.

2.4.3 Diagrama de paquetes

Los diagramas de paquetes expresan cómo está dividido el sistema en agrupaciones lógicas, mostrando las dependencias entre estas (15). Los paquetes permiten describir la arquitectura y el particionamiento de un sistema haciendo uso de la notación UML. Durante el desarrollo de

software resulta muy conveniente agrupar clases y ficheros por diferentes criterios que ayudarán a una fácil comprensión de la aplicación.

Seguidamente se muestra el diagrama de paquetes que contiene la estructura del módulo Información financiera (*ver Figura 2.5*), al cual pertenece el requisito Registrar información financiera.

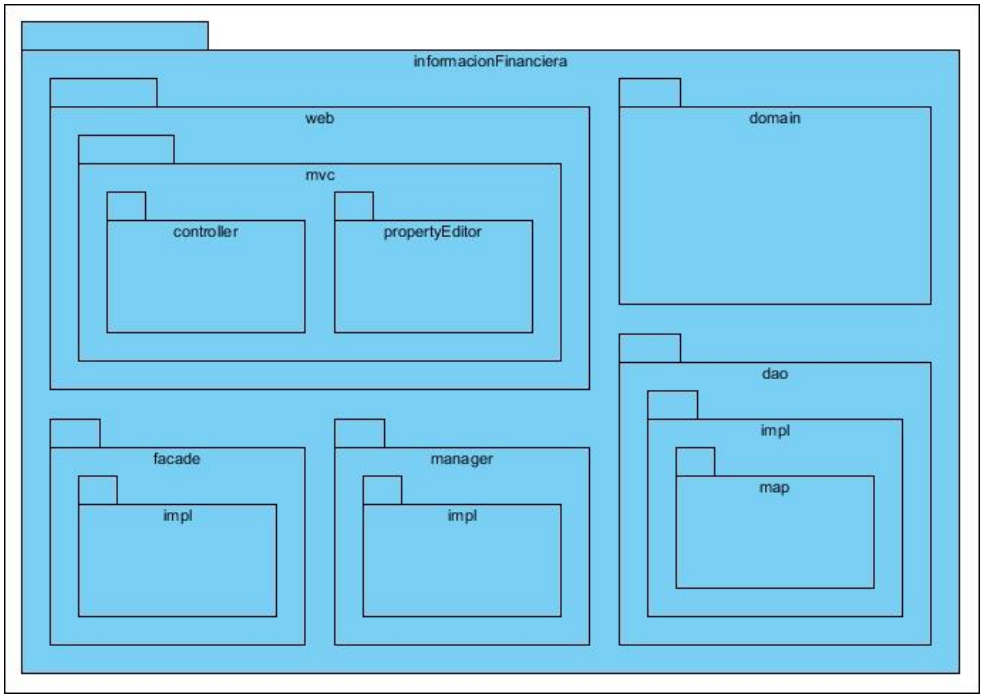

Figura 2.5. *Diagrama de paquetes del módulo Información financiera.*

2.4.3.1 Descripción de paquetes

Paquete 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.

Paquete mvc: en este sub-paquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring MVC.

Paquete controller: en este paquete se hallan las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.

Paquete propertyEditor: agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

Paquete facade: en este paquete se encuentran las interfaces y las implementaciones, encargadas de brindar las funcionalidades que serán usadas por la presentación, así como las que se consumen por otros módulos.

Paquete manager: en este paquete se hallan las interfaces y las implementaciones que comprenden la lógica de negocio que existe en el módulo, que serán brindadas a las capas superiores.

Paquete dao: es donde 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.

Paquete domain: es donde se localizan las clases relacionadas con el dominio del módulo en cuestión.

2.4.4 Diagrama de clases

El diagrama de clases del diseño es el artefacto que representa los conceptos en un dominio del problema. Este diagrama es utilizado durante la disciplina de Análisis y diseño y describe la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. Constituye el diseño conceptual de la información que se maneja en el sistema y los componentes que se encargan del funcionamiento.

El diagrama de clases del diseño perteneciente al módulo Información financiera, atendiendo a la arquitectura definida, puede ser consultado en el repositorio del proyecto Quarxo.

2.4.5 Diagramas de secuencia

Los diagramas de secuencia definen las acciones que se pueden ejecutar y muestran las interacciones ordenadas entre los objetos de un escenario. O sea, estos indicarán los módulos o clases que forman parte del sistema y las llamadas que se hacen en cada uno de ellos para realizar una tarea determinada. Cuando los requisitos a modelar poseen varios flujos distintos, es recomendado crear un diagrama de secuencia por cada escenario. Seguidamente se muestran los generados para el requisito estudiado (*ver Figura 2.6, Figura 2.7*).

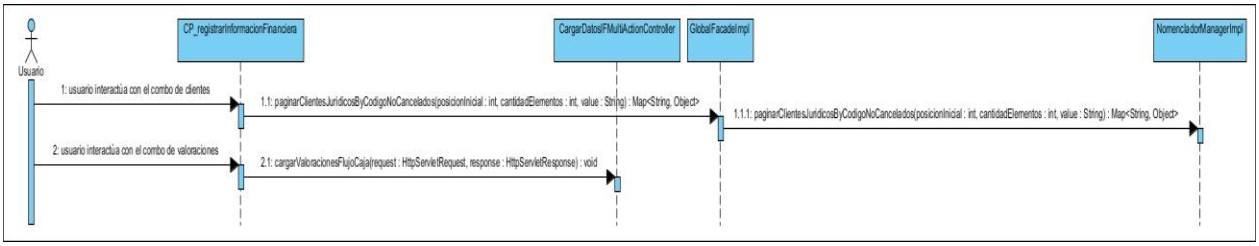

Figura 2.6. *Diagrama de secuencia del requisito Registrar información financiera (primer escenario).*

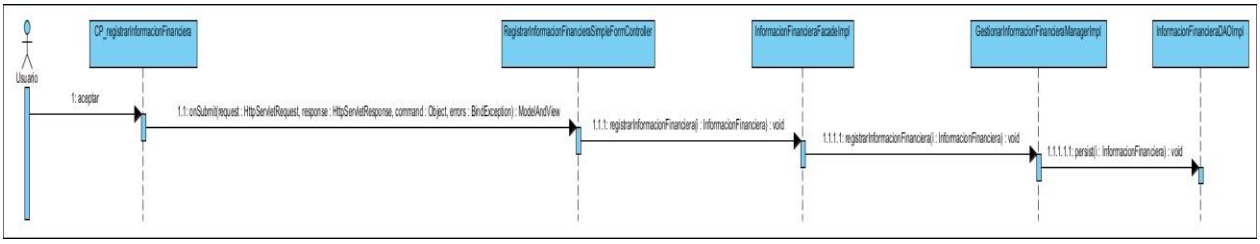

Figura 2.7. *Diagrama de secuencia del requisito Registrar información financiera (segundo escenario).*

2.4.6 Diseño de casos de pruebas

Los casos de pruebas verifican si los requisitos de una aplicación son completamente satisfactorios, utilizando para ello un conjunto de condiciones o variables. Para cada caso de prueba existe una entrada conocida y una salida esperada, las cuales son determinadas antes de que se ejecute la revisión. Para determinar que un requisito es completamente satisfactorio se pueden realizar muchos casos de prueba. Sin embargo, con el propósito de asegurar la revisión de todos los requisitos de una aplicación, debe haber al menos un caso de prueba para cada uno de ellos. La descripción de los casos de pruebas puede consultarse en el repositorio del proyecto Quarxo.

2.4.7 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, GOF y de acceso a datos que se exponen posteriormente.

Entre los patrones GRASP empleados están:

- **Controlador**: la clase controladora CargarDatosIFMultiActionController constituye un ejemplo de la aplicación de este patrón. La misma funciona como intermediaria entre la interfaz

de usuario y el negocio. Además, tiene la responsabilidad de contener los métodos que cargan los datos que serán mostrados al usuario.

- **Experto**: se evidencia en la asignación de las funcionalidades a las clases a partir de la información que manejan. Un ejemplo de la utilización del patrón se encuentra en la clase InformacionFinancieraDAO, la cual es responsable de efectuar las operaciones persistir, consultar y eliminar las informaciones financieras. Sobre este mismo principio se realiza el diseño de las restantes funcionalidades.
- **Alta cohesión**: este patrón fue utilizado en el diseño de manera general. Se agruparon las clases en dependencia de los requisitos del módulo a los que se les debía dar respuesta y 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. La interfaz InformacionFinancieraFacade con su respectiva clase de implementación, permite que la clase de la presentación CargarDatosIFMultiActionController, 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 InformacionFinancieraFacade y su implementación. Las mismas son 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 una información que se encuentra en la base de datos, esta es atendida primeramente por el *Controller*, luego por el *Facade*, el *Manager* y finalmente el *DAO*. Así se evidencia la utilización de dicho patrón.

Patrón de acceso a datos utilizado:

- **DAO**: se evidencia con la definición de la clase InformacionFinancieraDAO y haciendo uso de la misma a través del framework Hibernate para almacenar los datos correspondientes.

2.5 Validación del diseño

La validación del diseño comprende las actividades que tienen como objetivo comprobar si el software desarrollado se ajusta a los requisitos del cliente. Esta tarea se llevó a cabo haciendo uso de varias métricas.

2.5.1 Métricas para la evaluación del diseño

Una métrica es un instrumento de medición que permite evaluar el software al inicio del proceso. Persigue comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a 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 (31).

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

2.5.1.1 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 (34). 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. En la tabla que se muestra a continuación (*ver Tabla 2.2*) se observan los tipos de afectaciones que pueden tener los atributos al evaluar el diseño según la métrica utilizada.

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

Tabla 2.2. *Afectaciones en el diseño según la métrica TOC.*

A continuación se muestran los valores de umbrales tomados en cuenta para evaluar el diseño propuesto y la complejidad por cada uno de los atributos (*ver Tabla 2.3*).

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

Tabla 2.3. *Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica TOC.*

Como resultado de la evaluación de la métrica TOC (*ver Figura 2.8*) se obtuvo lo siguiente:

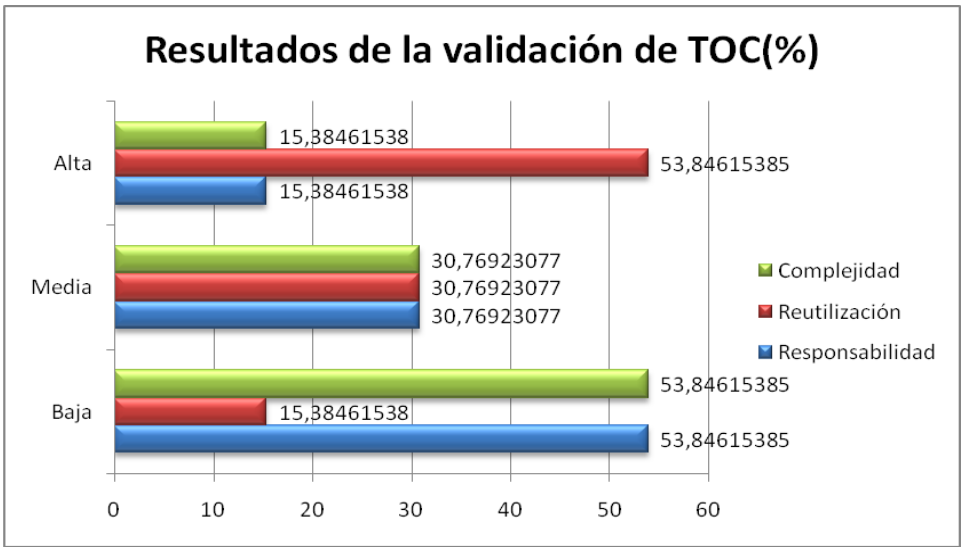

Figura 2.8. *Análisis de los resultados obtenidos en la evaluación de la métrica TOC.*

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 subsistema Análisis de Riesgo 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 (53%). Esto promueve el bajo acoplamiento entre las clases, facilitando la implementación y comprobación del desarrollador.

2.5.1.2 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 sean el acoplamiento, la complejidad de mantenimiento de una clase y la 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 (*ver Tabla 2.4*) son los siguientes:

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

Tabla 2.4. *Atributos de calidad y el modo en que son afectados.*

Para la evaluación de las clases fueron utilizados umbrales para el acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas (*ver Tabla 2.5*).

Atributo	Categoría	Criterio
Responsabilidad	Ninguna	0
	Baja	1
	Media	2
	Alta	> 2
Complejidad de mantenimiento	Baja	<= Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	> 2 * Promedio
Reutilización	Baja	> 2 * Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	<= Promedio
Cantidad de pruebas	Baja	<= Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	> 2 * Promedio

Tabla 2.5. *Atributos de calidad que afecta esta prueba.*

Como resultado de la evaluación de la métrica RC (*ver Figura 2.9*) se obtuvo lo siguiente:

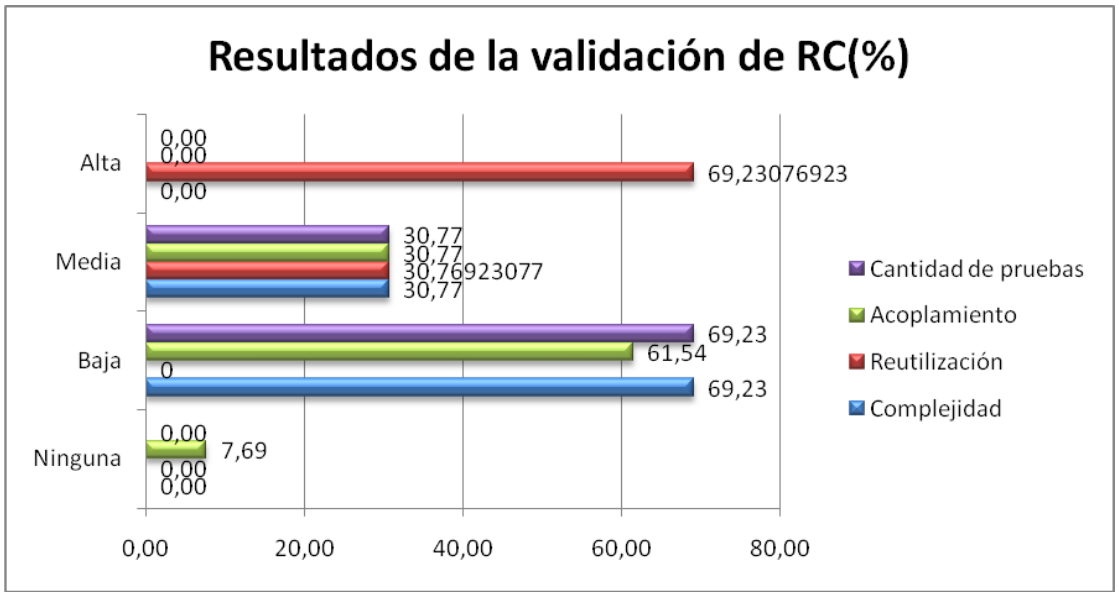

Figura 2.9. *Análisis de los resultados obtenidos en la evaluación de la métrica RC.*

De los resultados obtenidos en la evaluación de la métrica RC se puede deducir que el diseño del subsistema tiene una calidad aceptable, teniendo en cuenta que aproximadamente el 62% 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 (62%), así como la complejidad de mantenimiento (69%), e igualmente la cantidad de pruebas a realizar representan un bajo por ciento (69%). En consecuencia el grado de reusabilidad es mayor.

2.6 Conclusiones parciales

Luego del desarrollo del capítulo 2 se concluye lo siguiente:

- Como parte de la metodología del centro CEIGE, fue realizada la disciplina Modelado del Negocio, en la cual se comprendieron los procesos fundamentales relacionados con el análisis de riesgo de créditos en el Banco Nacional de Cuba. El análisis de esos procesos dio lugar a la captura de seis requisitos funcionales.
- Como parte de la disciplina Análisis y diseño, se realizó el modelo de diseño del módulo Información financiera que abarcó un grupo de artefactos, entre ellos: el diagrama de clases, el diagrama de paquetes, los diagramas de interacción y el modelo de datos. El modelo de diseño realizado constituirá la entrada principal a la disciplina Implementación.
- El diseño fue validado por medio de la aplicación de las métricas TOC y RC.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En este capítulo se muestra el modelo de implementación que pone en práctica el diseño de la solución realizado en el capítulo anterior, las pruebas realizadas al software que evalúan la calidad del sistema y se hace una valoración de las mismas según los resultados obtenidos. Se expone la nomenclatura usada tanto en el código como en los paquetes y clases de los módulos.

3.2 Estándares de codificación

Los estándares de codificación se definen por el equipo de desarrollo para lograr la 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 (35).

3.2.1 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.

El proyecto Quarxo decidió definir para la nomenclatura de las clases la notación PascalCasing, la cual define que los nombres deben comenzar por letra mayúscula, en caso de estar formado por palabras compuestas el inicio de cada palabra deberá tener mayúscula, se obvia el uso de artículos y se tiene en cuenta el tipo de clase que representa, entiéndase como tipo el rol que ellas desempeñan en el sistema (35).
Ejemplo: GestionarIFManager. En este caso el nombre de la clase está compuesto por cuatro 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**: a las clases incluidas en este paquete, después del nombre se le incorpora el nombre del controlador de Spring del cual hereda (NombreClaseNombreControladorClases).

Ejemplo: CargarDatosIFMultiActionController.

- **PropertyEditor**: a las clases que se encuentran dentro de este paquete después del nombre se le agrega la palabra PropertyEditor (NombreClasePropertyEditor).

Ejemplo: ClientePropertyEditor.

- **Facade**: a las clases incluidas en este paquete, después del nombre del módulo en cuestión se le agregan 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: InformacionFinancieraFacade, InformacionFinancieraFacadeImpl.

- **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: GestionarIFManager, GestionarIFManagerImpl.

- **Dao**: a 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 la notación CamelCasing, que es muy similar a PascalCasing con la excepción de que la letra inicial debe estar en minúscula.

Ejemplo: registrarInformacionFinanciera. Este nombre de método está compuesto por tres palabras, la primera todo en minúsculas y las otras dos iniciando con letra mayúscula. Lo mismo se aplica a los nombres de ficheros de código JavaScript, 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.

3.3 Modelo de implementación

El modelo de implementación describe la manera en que los elementos del modelo de diseño tales 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 (13).

3.3.1 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 (36).

El diagrama de componentes que se muestra a continuación (*ver Figura 3.1*) se ha elaborado de forma tal que muestre las relaciones existentes entre el subsistema Análisis de Riesgo y el resto de los componentes con los que se relaciona dentro del sistema.

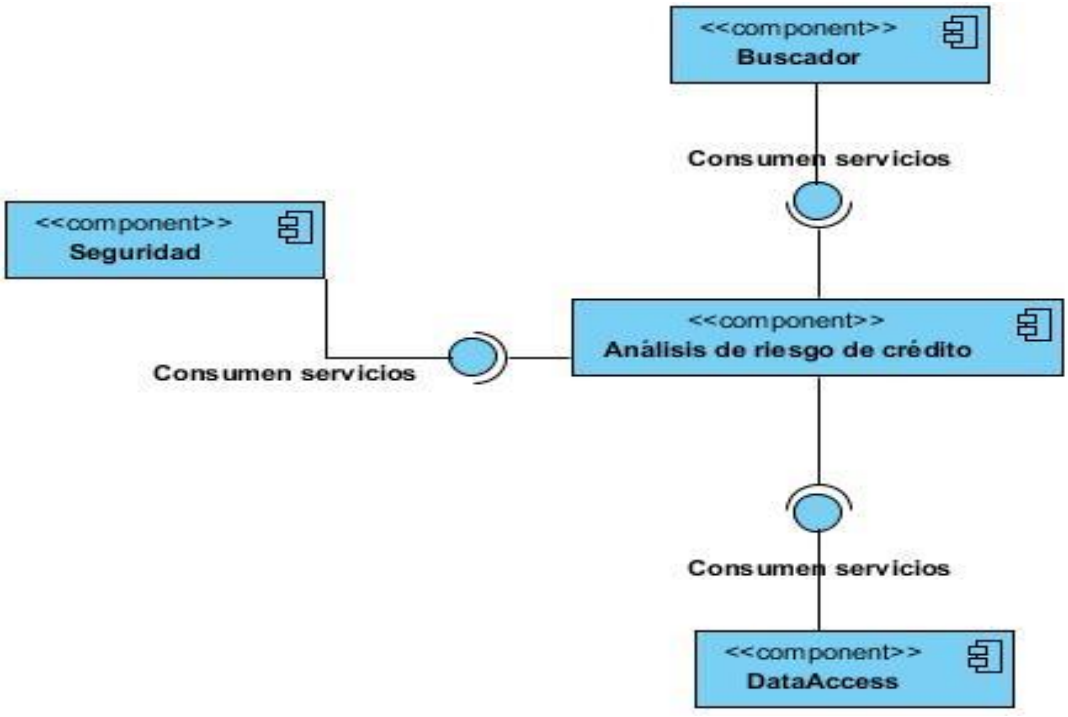

Figura 3.1. *Modelo de componentes correspondiente al subsistema Análisis de Riesgo.*

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 créditos.
- **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 subsistema Análisis de Riesgo se hace necesaria la búsqueda de informaciones financieras.
- **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 Descripción de las clases y funcionalidades

Teniendo en cuenta el diseño de las clases correspondientes al subsistema Análisis de Riesgo realizado, a continuación se describirán las clases, atributos y métodos más importantes para el subsistema desde el punto de vista funcional.

Se analizará la clase InformacionFinanciera (*ver Tabla 3.1*) que permite representar en forma de objeto los datos de una información financiera, la clase CargarDatosIFMultiActionController (*ver Tabla 3.2*) que es la encargada de cargar datos, calcular y atender algunas de las peticiones relacionadas con el módulo, y finalmente, la clase InformacionFinancieraDAOImpl (*ver Tabla 3.3*) encargada de realizar todas las operaciones a nivel de base de datos.

Nombre: InformacionFinanciera	
Tipo de clase: Modelo	
Atributo	**Tipo**
idInformacionFinanciera	Integer
oClienteJuridico	ClienteJuridico
balancesGenerales	java.util.List
estadosResultados	java.util.List
flujosCajas	java.util.List

Tabla 3.1. *Descripción de la clase InformacionFinanciera.*

Nombre: CargarDatosIFMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
informacionFinancieraFacade	InformacionFinancieraFacade
Para cada responsabilidad:	
eliminarInformacionFinanciera(HttpServletRequest request, HttpServletResponse response)	Se encarga de eliminar una información financiera dado su identificador.
cargarJsonIF(HttpServletRequest request, HttpServletResponse response)	Permite construir un JSONObject con los datos necesarios para mostrar en la vista.
cargarValoracionesFlujoCaja(HttpServletRequest request, HttpServletResponse response)	Permite cargar las distintas valoraciones del flujo de caja.
calcularIndicadores(String codReeup, BalanceGeneral bg)	Se encarga de calcular los indicadores en dependencia de la institución a la que pertenece el cliente.
existeInformacionFinanciera(HttpServletRequest request, HttpServletResponse response)	Permite conocer dado un identificador si existe la información financiera.

Tabla 3.2. *Descripción de la clase CargarDatosIFMultiActionController.*

Nombre: InformacionFinancieraDAOImpl	
Tipo de clase: Data Access Object	
Atributo	**Tipo**
Para cada responsabilidad:	
persist(InformacionFinanciera i)	Se encarga de persistir una información financiera en la base de datos.
findById(int id)	Se encarga de obtener de la base de datos la información financiera con el identificador correspondiente.
delete(InformacionFinanciera i)	Se encarga de eliminar la información financiera en la base de datos.

Tabla 3.3. *Descripción de la clase InformacionFinancieraDAOImpl.*

3.5 Validación de la solución

La calidad del 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 subsistema Análisis de Riesgo cumple con las expectativas del cliente.

3.5.1 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 subsistema 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 aplicaron 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.

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).

3.5.1.1 Pruebas 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. 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 la implementación, haciéndolo flexible y testeable (41).

A continuación se muestran las imágenes que corresponden a la realización de las pruebas de caja blanca aplicando el framework JUnit al método existeInformacionFinanciera () de la clase controladora CargarDatosIFMultiActionController, dentro del módulo Información financiera del subsistema Análisis de Riesgo (*ver Figura 3.2*).

```
public boolean existeInformacionFinanciera(HttpServletRequest request, HttpServletResponse response) {
    String detalles = "";
    Boolean existe = false;
    JSONObject json = new JSONObject();
    int idInformacionFinanciera = Integer.parseInt(request.getParameter("id"));
    InformacionFinanciera informacionFinanciera = informacionFinancieraFacade.getInformacionFinancieraById(idInformacionFinanciera);
    if (informacionFinanciera != null) {
        existe = true;
        detalles = "m0";
    } else {
        detalles = "m1";
    }
    json.put("detalles", detalles);
    try {
        ResponseUtil.escribirDatosEnElResponse(response, json);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return existe;
}
```

Figura 3.2. *Método existeInformacionFinanciera de la clase CargarDatosIFMultiActionController.*

Para realizar las pruebas a dicho método primeramente se crea un objeto de la clase donde se encuentra el método a probar y a continuación se da paso a crear tres mocks, debido a que el método recibe como parámetros dos objetos de tipo HttpServlet (request y response). Esto lo posibilita la librería EasyMock que trabaja en la creación de un proxy dinámico para dicho simulacro, el cual es controlado a través de un objeto de tipo MockControl (*ver Figura 3.3*).

```
public class CargarDatosIFMultiActionControllerJUnitTestCase extends TestCase {

    private CargarDatosIFMultiActionController cargarDatos;
    @SuppressWarnings({ "unchecked" })
    private MockControl controlHttpServlet;
    private HttpServletRequest mockHttpServletRequest;
    @SuppressWarnings({ "unchecked" })
    private MockControl controlHttpResponse;
    private HttpServletResponse mockHttpServletResponse;
    @SuppressWarnings({ "unchecked" })
    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;

    @Before
    public void setUp() throws Exception {
        cargarDatos = new CargarDatosIFMultiActionController();
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "classpath:cu/uci/finixubnc/junit/analisisderiesgo context.xml");
        GestionarInformacionFinancieraManagerImpl manager = (GestionarInformacionFinancieraManagerImpl) context
                .getBean("GestionarInformacionFinancieraManager");
        InformacionFinancieraFacadeImpl facade = new InformacionFinancieraFacadeImpl();
        facade.setGestionarInformacionFinancieraManager(manager);
        cargarDatos.setInformacionFinancieraFacade(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 3.3. *Declaración de la clase CargarDatosIFMultiActionControllerJUnitTestCase para la prueba.*

En el método setUp, de la propia clase, se inicializan las variables antes de cada prueba. Debido a la arquitectura del subsistema se debe crear un objeto por cada capa que haga énfasis en el método a probar. Estos pasos se van a configurar en el fichero analisisderiesgo-context.xml, el cual va a ser el encargado de declarar y mapear los objetos creados por cada nivel de la aplicación (*ver Figura 3.4*).

```
@After
public void tearDown() throws Exception {
    controlHttpServlet.verify();
}

@Test
public void testExisteInformacionFinanciera() {
    mockHttpServletRequest.getParameter("id");
    controlHttpServlet.setReturnValue("1");
    controlHttpServlet.replay();
    assertTrue(cargarDatos.existeInformacionFinanciera(mockHttpServletRequest, mockHttpServletResponse));
}
```

Figura 3.4. *Método tearDown y testExisteInformacionFinanciera de la clase CargarDatosIFMultiActionControllerJUnitTestCase.*

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. En este caso verifica los objetos que son pasados por el request de forma automática para todos los test definidos en la clase CargarDatosIFMultiActionControllerJUnitTestCase.

Durante la realización de la prueba al método, se preparan los parámetros de pruebas necesarios para el método existeInformacionFinanciera. Dichos parámetros serán pasados por el mockHttpServletRequest. Posteriormente se realiza la condición de prueba a través del método assertTrue el cual compara el valor true con la respuesta del método.

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 resultado favorable (*ver Figura 3.5*) para dicha prueba.

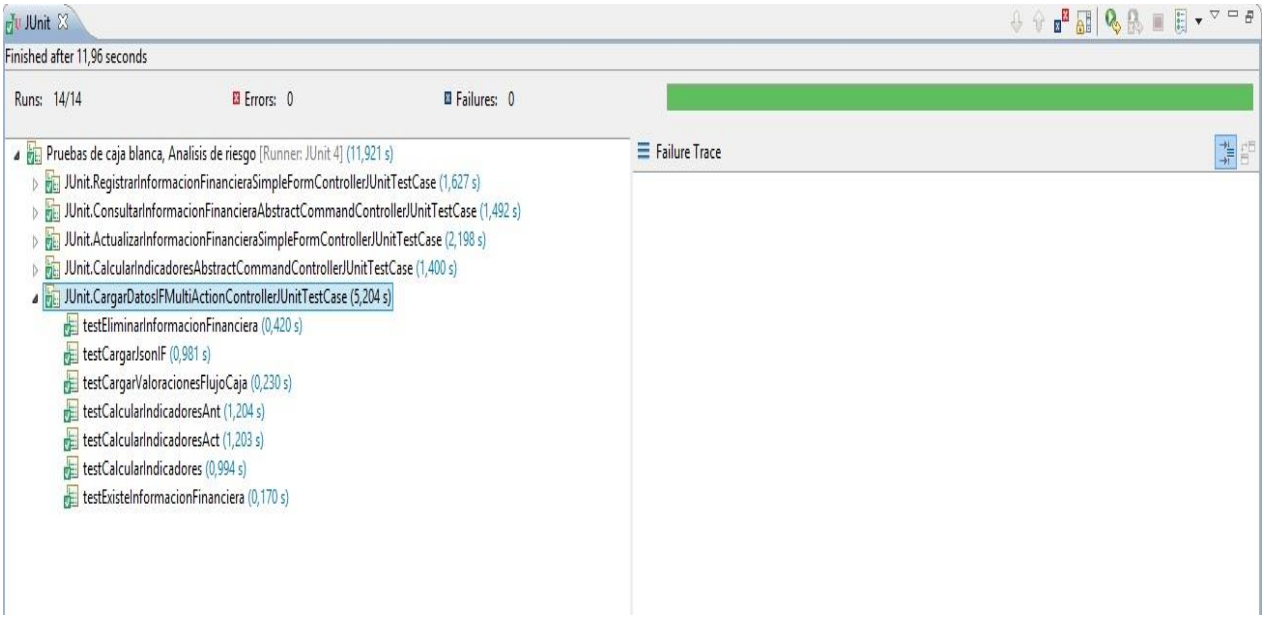

Figura 3.5. *Consola de JUnit con el resultado de la prueba realizada.*

3.5.1.2 Pruebas 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 Quarxo. Para ver el DCP del requisito funcional Registrar información financiera ver documento CIG-QF2-N-IFC-i5101 en el repositorio del proyecto Quarxo.

Para la aplicación de la prueba de caja negra se realizaron 3 iteraciones (*ver Tabla 3.4*), quedando validado el subsistema luego de que no se detectara ninguna no conformidad en la tercera iteración.

	Iteración 1	**Iteración 2**	**Iteración 3**
No Conformidades	9	4	0
Descripciones	Se detectaron cuatro errores ortográficos y cinco validaciones no realizadas.	Se detectaron dos mensajes informativos incorrectos, un error ortográfico y una validación no realizada.	No se detectaron errores.

Tabla 3.4. *Resumen de las iteraciones de las pruebas.*

3.6 Conclusiones parciales

Luego del desarrollo del capítulo 3 se puede concluir lo siguiente:

- El modelo de diseño realizado como parte de la disciplina previa a la implementación, constituyó a pesar de pequeños cambios que debieron hacerse, una guía fiel para el trabajo de los desarrolladores.
- El uso de la metodología y herramientas definidas en el proyecto Quarxo permitieron desarrollar con éxito la disciplina Implementación para el subsistema Análisis de Riesgo.
- La validación del subsistema realizada a través de pruebas de caja blanca y caja negra, mostraron un grupo reducido de no conformidades que fueron resueltas de inmediato.
- El subsistema desarrollado cumple con las expectativas del cliente y podrá ser implantado próximamente en el BNC.

CONCLUSIONES

Una vez terminado el presente trabajo de diploma se concluye lo siguiente:

- Del estudio realizado a sistemas nacionales e internacionales, el sistema del BICSA fue el que más aportó a la solución del presente trabajo. Del mismo se tomaron numerosos aspectos que fueron tenidos en cuenta para el desarrollo del subsistema Análisis de Riesgo del sistema Quarxo.
- La metodología, los lenguajes y las herramientas utilizados, permitieron desarrollar con éxito el objetivo de la investigación.
- Como parte de la metodología del centro CEIGE, fue realizada la disciplina Modelado del Negocio, en la cual se comprendieron los procesos fundamentales relacionados con el análisis de riesgo de créditos en el Banco Nacional de Cuba. El análisis de esos procesos dio lugar a seis requisitos funcionales.
- El diseño fue validado por medio de la aplicación de las métricas TOC y RC las cuales arrojaron resultados satisfactorios sobre el diseño realizado.
- La validación del subsistema a través de pruebas de caja blanca y caja negra, demostraron que el subsistema desarrollado cumple con las expectativas del cliente y podrá ser implantado próximamente en el BNC.

RECOMENDACIONES

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

- Seguir estudiando sistemas internacionales tales como el CreditScoring para futuras versiones del subsistema Análisis de Riesgo.
- Incorporar al subsistema Análisis de Riesgo módulos para la configuración de los indicadores contables y la gestión de los historiales crediticios de los clientes.
- Incorporar al subsistema Análisis de Riesgo técnicas de Inteligencia Artificial que apoyen al analista de riesgo de crédito en la toma de decisiones.

REFERENCIAS BIBLIOGRÁFICAS

1. **FOGACOOP.** *Riesgo de Crédito.* s.l. : http://www.fogacoop.gov.co/documentos/PRESENTACI%D3N%20RIESGO%20DE%20CR%C9DITO, 2011.

2. **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.* 2010.

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

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

5. **Gestiopolis.** *Aspectos básicos del análisis de riesgo.* s.l. : http://www.gestiopolis.com/recursos/documentos/fulldocs/fin/aspanalisiscreditos.htm. 29 de noviembre de 2011.

6. **Afirme, Banca.** *Administración Integral de riesgos. El banco de hoy.* s.l. : http://www.afirme.com.mx/Portal/VisualizadorContenido.do?archivoId=1573&menu=1., 2 de Diciembre de 2011.

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

8. **López Rodríguez, Ing. Yoan Antoniio, Saavedra Darias, Ing. Mavi y Osorio Turruelles, Ing. Yoel.** *Sistema para el análisis de riesgo de créditos (CREDIRI).* s.l. : http://uciencia.uci.cu/es/node/1056., 12 de Junio de 2012.

10. *Scalar Consulting.* s.l. : http://www.grupoescalar.com/software/riesgo_credito_scoring.htm., 15 de Noviembre de 2011.

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

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

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

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

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

16. **Desarrolloweb.com, Equipo.** *¿Qué es Java?* s.l. : http://www.desarrolloweb.com/articulos/497.php., 24 de Noviembre de 2011.

17. **Corporation., Oracle.** *Sun MicroSystems.* s.l. : http://java.sun.com/javaee/5/docs/tutorial/doc/., 28 de Noviembre de 2011.

18. **XHTML., http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.1. Manual de.** *Manual de XHTML.* s.l. : http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html., 24 de Noviembre de 2011.

19. **Rodríguez, José Antonio.** *MANUAL DE JAVASCRIPT.*

20. *Codebox. Glosario.* s.l. : http://www.codebox.es/glosario.

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

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

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

24. **Scribd.** *Scribd.* s.l. : http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE., 28 de Noviembre de 2011.

25. *Visual Paradigm.* s.l. : http://www.visual-paradigm.com., 28 de Noviembre de 2012.

26. *Control de versiones con Subversion.* s.l. : http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html.

27. **Foundation, Eclipse.** *Eclipsepedia.* s.l. : http://wiki.eclipse.org/Main_Page., 2 de Diciembre de 2011.

28. *Apache Tomcat.* s.l. : http://tomcat.apache.org/.

29. **Corporation., Microsoft.** *Microsoft SQL Server.* s.l. : http://www.microsoft.com/sqlserver/2005/en/us/product-information.aspx., 28 de Noviembre de 2011.

30. **Web., Desarrollo.** *Usabilidad y arquitectura del software.* s.l. : http://www.desarrolloweb.com/articulos/1622.php., 15 de Enero de 2012.

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

32. **Kendall, Kenneth E. y Kendall, Julie E.** *Análisis y diseño de sistemas. Sexta Edición. México : PEARSON EDUCACIÓN,.* s.l. : págs. 693-694, 2005. ISBN 970-26-0577-6.

33. *Definición de modelo de datos.* s.l. : http://definicion.de/modelo-de-datos/, 2012.

34. **Negro, Ing. Pablo Ariel.** *Umbrales para métricas orientada a objetos.* s.l. : http://caeti.uai.edu.ar/archivos/271_TESIS.PDF, 2008.

35. **Smania, Sofia y Duarte, Noelia.** *Tecnologías en Desarrollo de Software IDE - UTN.* s.l. : tp://subversion.assembla.com/svn/net-utn2008/trunk/TPs%201/08.309.TP1.Smania-Duarte.Convenciones/, 5 de Junio de 2012.].

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

37. **ENTIDADES, CENTRO DE INFORMATIZACIÓN DE LA GESTIÓN DE.** *MODELO DE DESARROLLO DE SOFTWARE.* La Habana : Subdirección de Producción : s.n., 2012.

38. **S. Pressman, Roger.** *Ingeniería del Software, un enfoque práctico.* s.l. : McGraw : s.n., 1997.

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

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. **Ramos Reyes, Jessica**. Informatica76Informática sin límiteshttp://informatica763.webnode.mx/news/actividad-6-objetos-de-flujo-bpm-business-process-model-and-notation-bpmn-/.

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 de crédito. Constituye la valoración final de este sobre el proceso de análisis de riesgo de créditos.

Información financiera: conjunto de datos que se utilizan para conocer el patrimonio o los resultados de la operación de algún negocio. Comprende al balance general, el estado de resultados y el flujo de caja.

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.

Flujo de caja: muestra la proyección del crédito solicitado al banco en las operaciones de entradas y salidas de la empresa.

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.