Universidad de las Ciencias Informáticas

Facultad 3

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

Título: ‘‘Desarrollo del módulo Autorizo de desembolso del sistema Quarxo’’.

Autor: Raidel Montero Alvarado.

Tutores

Yoan Antonio López Rodríguez.

Luis Carlos Guerra Rodríguez.

La Habana, junio de 2014

“Año 56 de la Revolución”

DECLARACIÓN DE AUTORÍA

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

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

Raidel Montero Alvarado

_____

Firma del Autor

Ing. Yoan Antonio López Rodríguez

_____

Firma del Tutor

Ing. Luis Carlos Guerra Rodríguez

_____

Firma del Tutor

DATOS DE CONTACTO

Ing. Yoan Antonio López Rodríguez
Ingeniero en Ciencias Informáticas, graduado en julio del 2008
Profesor Asistente
Dpto.: Soluciones Financieras y Aduanales
Centro: Informatización de Entidades
Facultad 3
Universidad de las Ciencias Informáticas (UCI)
Líneas investigativas en las que ha trabajado: Software de gestión y software educativo

Ing. Luis Carlos Guerra Rodríguez
Ingeniero en Ciencias Informáticas, graduado en julio del 2013
Dpto.: Soluciones Financieras y Aduanales
Centro: Informatización de Entidades
Facultad 3
Universidad de las Ciencias Informáticas (UCI)
Líneas investigativas en las que ha trabajado: Software de gestión

AGRADECIMIENTOS

DEDICATORIA

RESUMEN

La informatización del Sistema Bancario Cubano como parte del desarrollo tecnológico que precisa el país, demanda una elevada capacidad tecnológica y operativa, lo que trae consigo la utilización de herramientas computacionales para el procesamiento de la información. La Universidad de las Ciencias Informáticas ha desempeñado un papel protagónico en ese sentido, de manera que está inmersa en el desarrollo del sistema Quarxo, que responde de forma rápida y certera a la actividad contable y financiera que se lleva a cabo en el Banco Nacional de Cuba. Dentro de los procesos más importantes que no están automatizados en el BNC con la organización y calidad requerida, se encuentra la gestión de los autorizos de desembolsos. El presente trabajo de diploma comprende el Análisis, Diseño e Implementación del módulo Autorizo de desembolso. Con vistas a satisfacer al cliente, el contenido del trabajo incluye la caracterización de patrones de diseño, herramientas y tecnologías de la plataforma Java a utilizar por sus potencialidades como software libre para el desarrollo de aplicaciones web, la generación de los principales artefactos de salida de los flujos de trabajo del desarrollo de software involucrados y los elementos de validación de la solución propuesta. Como resultado se obtiene el módulo Autorizo de desembolso, el cual contribuye al aumento de la productividad de los trabajadores y a la disminución de gastos de recursos materiales en el Banco Nacional de Cuba.

PALABRAS CLAVES

Autorizo de desembolso, garantía, crédito, módulo.

ÍNDICE

ÍNDICE

ÍNDICE

INTRODUCCIÓN

La información es considerada la fuerza impulsora de la economía y la sociedad, de ahí que los avances en las tecnologías hayan propiciado un incremento de la productividad. La informatización de la economía y el papel dominante que las nuevas tecnologías han desempeñado, han producido un gran efecto estimulando nuevas inversiones productivas, una mayor utilización del capital y novedosas formas para incrementar la producción. Para poder ganar a la competencia los bancos deben mejorar la atención a los clientes a través de una gestión de datos más sofisticada. Con el objetivo de aumentar su eficiencia, el sector bancario debe proporcionar niveles superiores de servicio y mayor rapidez en adoptar los últimos avances tecnológicos, pensando siempre y en primer lugar en sus clientes.

La UCI[1] ha venido trabajando con el BNC[2] mediante un sistema llamado Quarxo, con el cual se logra, no solo mayor rapidez en los procesos, sino también, el almacenamiento de un conjunto de información estadística usada para la toma de decisiones y en la obtención de reportes. Quarxo está integrado por varios subsistemas dentro de los cuales se encuentra el subsistema Crédito. El dinamismo que exige el mundo actual para operar con los préstamos, se garantiza solamente con una gestión informatizada. El subsistema Créditos les permite a los usuarios gestionar tanto los préstamos que son recibidos por el banco como los que son concedidos a sus clientes. El subsistema se interrelaciona con otros subsistemas tales como: Contabilidad, permitiendo la contabilización en tiempo real de las transacciones; Mensajería, posibilitando que los mensajes se envíen automáticamente, entre otros. En estos momentos se ha solicitado mejorar la gestión de los procesos de autorización de desembolso que en el sistema Quarxo se realizan, ya que actualmente no es posible la gestión personalizada de esos procesos, lo cual provoca retrasos a dicha entidad.

Actualmente el sistema Quarxo cuenta con un módulo para la personalización de las garantías, sin embargo, una vez que se autoriza el desembolso de las mismas, es preciso que los usuarios del sistema realicen la transacción contable correspondiente, a través de la Transacción General, es decir sin la debida personalización del proceso tal como hoy existe para la mayoría de los procesos que se gestionan a través del sistema Quarxo. La realización de la transacción a través de la Transacción General requiere la entrada al sistema de casi toda la información que abarca una transacción contable, incluyendo los números de las

[1]Universidad de las Ciencias Informáticas.
[2]Banco Nacional de Cuba.

cuentas, lo cual trae consigo que los usuarios tengan que conocer esos números de cuentas, hecho que provoca a su vez, una frecuente entrada de errores al sistema. Asimismo, el uso de la Transacción General para la gestión de un determinado proceso, le provoca al BNC otras deficiencias tales como: el retraso, por la entrada de gran volumen de información de forma repetida y el no almacenamiento de la información estadística que acompaña a la operación, en este caso, al autorizo de desembolso.

Por los motivos antes expuestos se plantea como **problema a resolver:** No se cuenta con una gestión personalizada del proceso autorizo de desembolso lo cual provoca deficiencias al BNC.

Como **objeto de estudio** se tiene: La informatización del proceso autorizo de desembolso en las entidades financieras bancarias.

Dentro del objeto de estudio se enmarca el siguiente **campo de acción**: la gestión del proceso autorizo de desembolso en el BNC.

Se plantea como **objetivo general** desarrollar un módulo que permita la gestión personalizada del proceso autorización de desembolso, para contribuir a mejorar dicho proceso en el BNC.

Como **idea a defender** de la investigación se tiene que con el desarrollo del módulo Autorizo de desembolso para el sistema Quarxo, se permitirá la gestión personalizada del proceso, la cual contribuirá a mejorar el proceso en el BNC.

Se definen como objetivos específicos:

1. Elaborar el marco teórico de la investigación para la formulación de los principales conceptos y la propuesta del módulo Autorizo de desembolso del sistema Quarxo.
2. Realizar el modelado del negocio, el análisis y el diseño del módulo Autorizo de desembolso.
3. Implementar las funcionalidades del módulo Autorizo de desembolso bajo las tecnologías y herramientas definidas en el proyecto Quarxo para satisfacer las necesidades del cliente.
4. Validar la solución propuesta para asegurar la calidad del software y la aceptación del cliente.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de investigación:**

1. Analizar la metodología, herramientas y tecnologías definidas en el proyecto Quarxo para el desarrollo.
2. Generación del marco teórico-conceptual a partir del análisis de sistemas que permitan la gestión de garantías.
3. Realización del Modelado del negocio del proceso autorizo de desembolso.
4. Definición de los requerimientos funcionales para el desarrollo del módulo Autorizo de desembolso.
5. Realización del diseño del módulo Autorizo de desembolso.
6. Validación del diseño del módulo Autorizo de desembolso.
7. Implementación del módulo Autorizo de desembolso.
8. Validación de la solución.

Para dar cumplimiento a las tareas anteriores se aplicaron los métodos de investigación científicos que a continuación se mencionan:

Métodos Teóricos: Los métodos teóricos permiten revelar las relaciones esenciales del objeto de investigación que no son observables directamente. Posibilitan la interpretación conceptual de los datos empíricos encontrados, además de explicar los hechos y profundizar en las cualidades fundamentales de los procesos, hechos y fenómenos (1).

1. Método Histórico – Lógico: utilizado para la realización de un estudio detallado de los sistemas contables que gestionan garantías a nivel nacional e internacional.
2. Método Analítico – Sintético: empleado en la realización del estudio teórico de la investigación y el análisis previo sobre los procesos de garantías, haciendo énfasis en el autorizo de desembolso de las garantías.
3. Método de Modelación: utilizado para la creación del modelo de diseño y el modelo de componentes.

Métodos Empíricos: los métodos empíricos revelan y explican las características fundamentales del objeto a través de procedimientos prácticos (1).

1. Entrevistas: se utilizó con el fin de obtener información, para ello fue preciso establecer diálogos planificados con los especialistas del BNC en aras de obtener los requisitos funcionales.

El presente trabajo está estructurado en tres capítulos. A continuación se brinda una breve reseña de cada uno para un mejor entendimiento del documento:

Capítulo 1. Fundamentación Teórica: se hace alusión a los principales conceptos para lograr un buen entendimiento del proceso autorizo de desembolso, se abordan sistemas contables para entidades bancarias en el mundo en los cuales se gestionen las garantías y sus desembolsos. Se hace un estudio de las técnicas, metodologías y herramientas definidas en el proceso Quarxo que se usarán en el desarrollo de la solución y finalmente se brindan las conclusiones del capítulo.

Capítulo 2. Análisis y diseño de la solución: Se centra fundamentalmente en el análisis y diseño de la solución, se caracterizan los patrones de diseño en la búsqueda de soluciones a problemas comunes del desarrollo para implementar las funcionalidades, se modelan los procesos de negocios, se realiza la especificación de requisitos, se describen la arquitectura que presenta el sistema y el modelo de diseño. Se exponen los artefactos generados tales como: los diagramas de clases del diseño, los diagramas de paquetes, los diagramas de secuencias y de flujo para conformar el modelo de diseño, entre otros. Por último se valida el diseño a partir de métricas.

Capítulo 3. Implementación y validación: Se enfoca directamente en la construcción de la solución explicando los aspectos principales de la implementación y la validación de los requisitos propuestos.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se abordan los conceptos de autorizo de desembolso necesarios para la comprensión del negocio, haciendo énfasis en los tipos, funcionamientos y las partes que intervienen en este proceso. De igual forma se realiza un estudio de los sistemas contables que operan los procesos de garantías con énfasis en el estudio de la manera en que gestionan los desembolsos de las mismas. Por último se especifica la metodología y se describen las herramientas, las tecnologías y los lenguajes que se utilizarán en el desarrollo del producto y que se encuentran definidos en el proyecto Quarxo. Para facilitar la comprensión del trabajo se deben entender primeramente los términos que se emplean en las instituciones bancarias, motivo por el cual se especifican algunos conceptos importantes tratados durante la investigación.

1.2 Marco Teórico

En esta sección se explican algunos aspectos fundamentales para la comprensión del contexto de la investigación. Inicialmente se abordan conceptos relacionados con la actividad bancaria de interés para la investigación.

1.2.1 Sistema bancario

El sistema bancario, o también conocido como banca, es el conjunto de entidades financieras que operan dentro de la misma economía. Según la literatura revisada (2) dentro de un sistema bancario se puede identificar distintos tipos de bancos, estos son:

- Bancos corrientes: son las instituciones financieras que ofrecen servicios de ahorro y disposición de cheques para personas físicas. También ofrecen préstamos y pueden ser vistos como intermediarios entre personas con excedentes de dinero y personas con necesidades de dinero.
- Bancos prestamistas especializados: son instituciones que otorgan préstamos o créditos con fines específicos, como pueden ser los Bancos Hipotecarios, entre otros.
- Bancos de Emisión: se les denomina así a los Bancos Centrales o Bancos Nacionales, son las únicas instituciones bancarias con la función de emitir dinero ya sea en papel o en moneda.

El sistema bancario se compone finalmente de los tres tipos de bancos vistos anteriormente, los cuales participan dentro de un mismo mercado, cada uno cumpliendo funciones distintas para atender las necesidades financieras de las personas físicas y jurídicas de la economía (2).

En sentido general los sistemas bancarios constituyen una estructura organizacional dentro de la cual se mueve el conjunto de instituciones bancarias, respondiendo a las directrices que le marca la autoridad superior; incluye la banca nacional, banca central, banca privada y mixta.

El sistema bancario cubano está encabezado por el Banco Central de Cuba y constituido por 9 bancos comerciales, 15 instituciones financieras no bancarias, 11 oficinas de representación de bancos extranjeros en Cuba y 4 oficinas de representación de instituciones financieras no bancarias (3).

Entre los bancos cubanos se puede citar el BNC, el cual una vez liberado de las funciones de banca central y de rector del sistema bancario, continúa existiendo con el carácter de banco comercial, autorizado a ejercer funciones inherentes a la banca universal, teniendo además la función de registro, control, servicio y atención de la deuda externa que el Estado y el propio banco han contraído con acreedores extranjeros con la garantía del Estado. (3)

A continuación se abordan conceptos relacionados con la actividad bancaria desarrollada en el BNC.

1.2.2 Crédito

Es una operación o transacción de riesgo en la que el acreedor (prestamista) confía a cambio de una Garantía en el tomador del crédito o deudor (prestatario), con la seguridad que este último cumplirá en el futuro con sus obligaciones de pagar el capital recibido (amortización de la deuda), más los intereses pactados tácitamente (servicio de la deuda) (2).

Las Líneas de crédito son un límite máximo al cual puede ascender el préstamo dentro del término de vencimiento acordado, es decir, es un monto determinado de dinero (fondo) que un banco u otro tipo de organización financiera ponen a disposición de una persona o una empresa. Es un contrato por el que una entidad se compromete a facilitar crédito a un cliente hasta un límite determinado (3). Durante el período de vigencia de la Línea de crédito, el prestatario puede disponer del mismo automáticamente. El objetivo más

importante de esa herramienta financiera es que el cliente cuente con estos fondos para realizar o cancelar otras operaciones.

1.2.3 Préstamo

Todo préstamo se efectúa entre un prestamista, quien da a préstamo el dinero, y un prestatario, quien lo recibe, originando una deuda de este último ante el primero. Los bancos y otras instituciones financieras asumen generalmente el papel de prestamistas, captando recursos que luego ofrecen a los interesados. También pueden prestarse bienes físicos, aunque en este caso suele hablarse por lo regular de un contrato de arrendamiento, donde el pago del alquiler correspondiente sustituye a los intereses (4).

1.2.4 Cartas de crédito

Las cartas de crédito son un instrumento de pago, sujeto a regulaciones internacionales, mediante el cual un banco (Banco Emisor) obrando por solicitud y conformidad con las instrucciones de un cliente (ordenante) debe hacer un pago a un tercero (beneficiario) contra la entrega de los documentos exigidos, siempre y cuando se cumplan los términos y condiciones del crédito. Este instrumento es uno de los documentos más sencillos en su forma y de los más complejos en cuanto a su contenido. Llamada también “Crédito Comercial”, “Crédito Documentario”, y en algunas ocasiones simplemente crédito.

1.2.5 Garantías

Las Garantías Bancarias son aquellas garantías que otorga una entidad bancaria para la seguridad de una obligación ajena. Suelen solicitarse por los clientes a favor de sus acreedores y por exigencias de estos últimos. Es el contrato por medio del cual un banco, a partir de la solicitud efectuada por un ordenante, se compromete irrevocablemente, a pagar una suma de dinero previamente establecida a un tercero (beneficiario) ante la simple solicitud que este haga al tiempo que adjunte ciertos documentos mínimos y previamente determinados en los cuales se certifica el incumplimiento de la obligación u obligaciones que surgen de un contrato base celebrado entre este y el ordenante (4).

1.2.6 Autorizo de desembolso

A la acción de utilizar la garantía emitida en el banco se le conoce como autorizo de desembolso. Generalmente en las negociaciones de carácter oficial en las que el estado respaldo una parte o el total de

los fondos de las operaciones realizadas por las empresas nacionales, se emite desde el inicio una garantía que luego es pagada por el banco a favor del beneficiario de la operación. El desembolso de una garantía puede darse de una sola vez, por el monto total de apertura o en diferentes cuotas, según lo pactado en el contrato de la garantía que incluye los respaldos de la misma. Se conoce como el documento que se emite por el Banco Nacional de Cuba autorizando a un banco extranjero a pagar a su cliente por los bienes y servicios brindados por éste y reconocidos por el cliente cubano.

1.3 Sistemas informáticos bancarios

Para el estudio de los sistemas informáticos 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 si permiten la gestión o no de las garantías bancarias. A partir de esos indicadores y persiguiendo extraer información valiosa para el desarrollo del módulo de la presente investigación, se hace un estudio de diferentes sistemas. Es importante mencionar que los sistemas líderes para la gestión de garantías, donde se incluyen los estudiados, son estadounidenses y propietarios, con lo que eso significa para Cuba. A continuación se presentan las valoraciones de los sistemas estudiados:

Integrated Banking System (e-IBS®)

Es un sistema bancario integrado, diseñado por Datapro, compañía americana de software bancario. Este sistema cuenta con más de 140 instalaciones en 27 países, de las cuales 44 se encuentran en el área de Centroamérica y Panamá. El sistema permite a sus usuarios la integración completa de sus operaciones, el control total del proceso y provee elementos dinámicos para la toma de decisiones administrativas. Para el óptimo manejo de Líneas de crédito dentro del sistema, el módulo de Líneas de Crédito del e-IBS® está compuesto por 4 opciones de menú: Nuevo/Mantenimiento, Consulta por Cliente, Consulta por Tipo de Líneas y Aprobación (40).

El módulo Garantías del e-IBS® es utilizado para crear nuevas Garantías dentro del sistema. Le permite a un usuario autorizado actualizar, aprobar o rechazar las Garantías abiertas y realizar la consulta de la información de las Garantías existentes y disponibles para el cliente. Este módulo está compuesto por cinco opciones de menú: Nuevo, Mantenimiento, Aprobación, Consulta y Asignación (41).

BANTOTAL

Bantotal es una solución uruguaya que comprende el procesamiento integral de todas las operativas de la Entidad Financiera (Core Bancario – Core Banking Solution). Está conformado por Sistemas Funcionales, orientados al procesamiento de los productos y servicios bancarios clásicos así como por Sistemas Núcleo, que constituyen la infraestructura de servicios que utilizan todos los Sistemas Bantotal (tanto Core como especializados). Dentro de sus principales sistemas funcionales se encuentran: el sistema de Líneas de Crédito y el sistema de Garantías.

ORION

Sistema Financiero diseñado para operar en Instituciones Financieras dedicadas al otorgamiento de crédito y captación de recursos del público. Especializado en la concesión de Líneas de crédito. Cuenta con un módulo de Garantías permitiendo la parametrización por clases de garantías y tipos de acuerdo a la Superintendencia de Bancos según necesidades y políticas de la institución financiera. Permite el manejo de seguro, cobertura y avalúos, permite el ingreso de detalles de las Garantías, valores y cantidades.

Resultados de la valoración de los sistemas bancarios

A pesar de que el negocio del BNC difiere del negocio de los bancos donde se utilizan los sistemas estudiados, presenta algunas similitudes. Hubiese sido factible contar con la documentación relacionada a los requisitos en cuanto a su definición y desarrollo lo cual no fue posible debido a que son soluciones propietarias. Sin embargo el estudio de los sistemas ofreció un aporte relacionado con la arquitectura de la información del sistema Quarxo, ya que permitió ubicar el módulo Autorizo de desembolso en el subsistema Créditos de igual manera que los módulos Préstamos, Garantías y Líneas de Créditos y no en el subsistema Cartas de Créditos como inicialmente se pensaba hacer.

1.4 Metodología, herramientas y tecnologías a utilizar

Para el desarrollo de esta investigación se hace necesaria la descripción de un conjunto de tecnologías y herramientas, las mismas son definidas por el proyecto al cual tributa esta investigación. Además, cabe destacar que las tecnologías que se proponen como solución en la investigación, constituye parte del mejoramiento al cual está sometido el sistema de Gestión Bancaria Quarxo, por lo que es necesaria la integración de los resultados de esta investigación con el sistema en un futuro cercano.

1.4.1 Metodología de desarrollo de software

Una metodología para el desarrollo de software es un conjunto de filosofías, frases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas informáticos (16).

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que se cumpla el objetivo por el cual fue creado.

Una metodología de desarrollo de software en general constituye un marco de trabajo basado en un conjunto de modelos, herramientas y métodos aplicados a la Ingeniería de Software usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Actualmente no existe una metodología universal que le haga frente con éxito a cualquier proyecto de desarrollo de software, puesto que para la selección de la metodología se ha de tener en cuenta las características del equipo de desarrollo. Es muy importante para lograr un producto con calidad escoger bien la metodología.

El centro CEIGE ha definido como metodología de desarrollo de software para sus aplicaciones el Modelo de Desarrollo de Software v1.1atendiendo a las características de los proyectos del centro y las áreas de procesos del nivel dos de CMMI (17). 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.

1.4.1.1 Modelo de desarrollo de software V1.1

El modelo de desarrollo de software es una guía o modelo estandarizado, que establece las distintas fases para el desarrollo de software en los proyectos del centro, el conjunto de procesos y productos de trabajo a generar en cada una de ellas teniendo en cuenta las áreas de proceso de CMMI nivel 2.

Características del Modelo de desarrollo:

- 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.
- 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.
- 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.

Descripción de las fases del ciclo de vida de los proyectos:

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

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 elabora la arquitectura y el diseño, se implementa y se libera el producto.

Figura 1.1. Desglose por fases del ciclo de vida del proyecto Quarxo

De los artefactos que propone el modelo de desarrollo en sus disciplinas se realizarán los siguientes:

Modelado de negocio:

- Descripción de procesos de negocio.
- Modelo conceptual.
- Reglas del negocio.

Requisitos:

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

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.

Implementación:

- Modelo de componentes.

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.

Para el desarrollo de las primeras disciplinas del ciclo de vida del proyecto: Modelado de Negocio y Requisitos, es fundamental la interacción con los clientes y expertos del negocio, para ello se realiza la Elicitación de requisitos (ER) con el objetivo de crear las primeras relaciones con el equipo de desarrollo y así obtener la información necesaria para determinar las necesidades reales de la organización.

Con el objetivo de lograr una identificación precisa de los requisitos se han desarrollado diferentes técnicas que pueden ser utilizadas en forma combinada para aumentar la calidad de la información que fluye de los clientes a los desarrolladores.

- Modelado de Negocio: en el modelo de negocio se describen los procesos del negocio y es fundamental para la comprensión general de los mismos.
- Entrevistas: Es la más utilizada, ya que es la forma natural de entendimiento de los humanos. Es importante conocer el vocabulario del dominio del problema para lograr un buen entendimiento con el cliente. En la presente investigación se utilizaron las entrevistas informales y guiadas por expertos del equipo de desarrollo.

Para validar que los requisitos identificados cumplen con las necesidades del cliente se realiza la validación de los requisitos (VRE)

- Revisiones: está técnica consiste en la lectura y corrección de la completa documentación o modelado de la definición de requisitos. Con ello solamente se puede validar la correcta interpretación de la información transmitida.

- Auditorías: esta técnica consiste en la revisión de la documentación, controlando los resultados contra una lista de chequeo predefinida o definida a comienzos del proceso, es decir sólo una muestra es revisada.

- Prototipos: algunas propuestas sobre esta técnica señalan que se basa en obtener de la definición de requisitos prototipos, que a pesar de no tener la totalidad de la funcionalidad del sistema, permitan al usuario hacerse una idea de la estructura de la interfaz del sistema con el usuario. Esta técnica tiene como problema que el usuario debe entender que lo que está viendo es un prototipo y no el sistema final.

1.4.2 Lenguajes

Un lenguaje informático es un idioma usado por ordenadores por lo que en ocasiones se refiere como lenguaje de ordenadores, diseñados para transmitir información mediante equipos de cómputo. La clasificación de lenguaje informático engloba diferentes tipos tales como los lenguajes de programación, modelado, consulta, sonido, gráfico, marcas y pseudocódigos. En esta sección se describirán los diferentes lenguajes de modelado y de programación utilizados en el desarrollo del producto.

1.4.2.1 Lenguaje de modelado y notación

UML es el estándar industrial de mayor utilización en la actualidad, patrocinado por el Grupo de Gestión de Objetos (OMG, del inglés Object Management Group), usado fundamentalmente para visualizar, especificar, construir y documentar los artefactos de un sistema de software (4). El mismo incluye reglas para describir un plano del sistema (modelo), así como aspectos conceptuales tales como: procesos de negocio y funciones del sistema, aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. UML utiliza herramientas de modelado visual facilitando la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. Ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. En resumen, ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software (19).

BPMN[3] es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. 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, del inglés Business Process Diagram*) (21). 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.

1.4.2.2 Lenguajes del lado del servidor

Se les clasifica así a los lenguajes que son reconocidos, interpretados y ejecutados en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente.

Existen diferentes lenguajes de gran popularidad por la cantidad de software que se han desarrollado sobre ellos y los resultados obtenidos con las aplicaciones, algunos de estos son: C# desarrollado y estandarizado por Microsoft que soporta el paradigma orientado a objetos, PHP diseñado para la creación de páginas web dinámicas siendo además multiparadigma y Java lenguaje de código abierto y multiplataforma. A continuación se presentan las principales características del lenguaje del lado del servidor que será usado en el desarrollo de la propuesta de solución.

Java 6 es un lenguaje de programación orientado a objetos. Inspirado en el lenguaje C++, proyectado con la finalidad de obtener grandes productos en entornos empresariales. Es simple y portátil sobre diferentes plataformas y sistemas operativos, lo que significa que los programas Java pueden ser ejecutados sobre cualquier computadora en la cual esté instalada la máquina virtual. Es usado en la web para la carga y ejecución de programas en el ordenador cliente (4). Java es un lenguaje de programación orientado a objetos, robusto y fácil de aprender por la gran documentación con que cuenta, diseñado para crear software altamente fiable, permite la codificación de la propuesta de solución de una manera rápida y flexible, al integrarse con los

[3] *Business Process Management Notation. Notación para el modelado de procesos de negocio.*

diferentes marcos de trabajos de desarrollo, permite programar páginas web dinámicas con accesos a bases de datos utilizando XML con cualquier tipo de conexión de red entre cualquier sistema.
Java fue seleccionado por el equipo de arquitectura para su empleo en el desarrollo atendiendo a las ventajas que este ofrece y al conocimiento y preparación del equipo de desarrollo con que se contaba al iniciar el proyecto, basándose además en los resultados obtenidos por el proyecto SIGEP[4], el cual usó este lenguaje y desarrolló varias librerías para el mismo con el objetivo de llevar a cabo un desarrollo mucho más rápido logrando montar una arquitectura robusta que se adaptaba a las necesidades del sistema Quarxo.

1.4.2.3 Lenguajes del lado del cliente

Un lenguaje del lado cliente es totalmente independiente del servidor, lo cual permite que la página pueda ser albergada en cualquier sitio, para su correcta visualización es necesario que la computadora cliente tenga instalados los plugins adecuados. Una de las desventajas de los lenguajes del lado del cliente es que el código tanto del hipertexto como de los scripts, es accesible a cualquiera y ello puede afectar a la seguridad. Seguidamente se presentan las principales características de los lenguajes del lado del cliente que serán usados en el desarrollo de la propuesta de solución.

Javascript es un lenguaje de programación que realiza acciones dentro del ámbito de una página web. Es compatible con la mayoría de los navegadores modernos. El código es ejecutado en el cliente por lo que el navegador (browser) se encarga de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades. Es un lenguaje interpretado, que no requiere compilación y es utilizado generalmente para la construcción de páginas web en combinación con el HTML, para el desarrollo de aplicaciones cliente-servidor dentro de Internet.

JSP[5] es un lenguaje para la creación de sitios web dinámicos, orientado a desarrollar páginas web en Java y es multiplataforma. Desarrollado por Sun Microsystems creado para programar aplicaciones web potentes. Posee un motor de páginas basado en los servlets de Java y necesita tener instalado un servidor Tomcat.

[4] Sistema de Gestión Penitenciaria
[5] *Java Server Page*

En este caso se definió en el proyecto el uso del 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.

1.4.3 Herramientas

En informática, las herramientas son un conjunto de programas que se usan para crear un determinado producto, normalmente otro programa o sistema informático. Resulta fundamental después de haber definido la metodología que guiará el proceso de desarrollo del software y los lenguajes que se utilizarán para la realización del diseño e implementación del sistema, definir las herramientas adecuadas sobre la que se deberá desarrollar el producto.

1.4.3.1 Herramientas de modelado CASE

Las herramientas CASE son un complemento de la caja de herramientas del ingeniero del software. CASE proporciona al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería. Las herramientas CASE ayudan a asegurar la calidad de un producto desde su diseño antes de construirlo (19). La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Existen varias herramientas que permiten una modelación con calidad, dentro de las más usadas se encuentra el Visual Paradigm que contiene características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar UML. A continuación se presentan las principales características del Visual Paradigm, herramienta que será usada en el modelado de la solución.

Visual Paradigm para UML v8.0 es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Ayuda a una rápida construcción de aplicaciones de calidad y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (19). Además es multiplataforma. Luego de un análisis realizado por parte de la dirección del proyecto sobre las posibles CASE a utilizar se decidió llevar a cabo el modelado del sistema con el Visual Paradigm. La herramienta se seleccionó por las características que presenta y las facilidades que brinda al integrarse con la notación BPMN y UML.

1.4.3.2 Entorno de desarrollo integrado

El entorno de desarrollo integrado (IDE, por sus siglas en inglés Integrated Development Environment) es un programa compuesto por un conjunto de herramientas para un programador. El mismo puede dedicarse a un solo lenguaje de programación o a varios. Además puede formar parte de aplicaciones existentes o ser una aplicación por sí solo. Estos sistemas informáticos han sido empaquetados como programas de aplicaciones. A continuación se presentan algunas características del IDE que se utilizará en el desarrollo de la aplicación.

Eclipse 3.5 JEE GALILEO IDE: es un entorno de desarrollo integrado, de código abierto y multiplataforma. Fue creado por la IBM (en inglés: International Business Machines) bajo la filosofía de software libre. Aunque Eclipse se usa como IDE para java, la arquitectura de plugins que posee permite integrar diversos lenguajes sobre un mismo IDE, además de introducir otras aplicaciones que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías, entre otros.

Plataforma J2EE: (conocido del inglés: Java 2 Platform, Enterprise Edition) define un estándar para el desarrollo de aplicaciones empresariales multicapa. La plataforma J2EE ofrece mejores perspectivas de desarrollo al basar su arquitectura en productos de software libre. J2EE permite soporte de múltiples sistemas operativos, marcos de trabajos y patrones de programación que permiten responder de una forma robusta y flexible a todas las demandas de este tipo de aplicaciones.

Después de un análisis se decidió por la dirección del proyecto en un inicio el uso de este IDE sobre la plataforma J2EE por las características que presenta y la experiencia de los desarrolladores con los mismos, comprobándose que en su mayoría usaban Eclipse.

1.4.3.3 Contenedor web

Como soporte para el funcionamiento de las aplicaciones desarrolladas en Java existen varios entornos de ejecución tales como: Jboss Server, que combina una arquitectura orientada a servicios revolucionaria con una licencia de código abierto; GlassFish, que implementa la plataforma J2EE y soporta las últimas versiones de tecnologías tales como: JSP, JSF, servlets, arquitectura Java para enlaces XML y Apache Tomcat, el cual no es considerado como un servidor de aplicaciones, sino como un contenedor de servlets, pero es capaz de soportar la mayoría de las aplicaciones desarrolladas en dicha tecnología. A continuación se describe brevemente el servidor Apache Tomcat que se empleará en la presente solución.

Apache Tomcat 6.0.35 es un contenedor web basado en el lenguaje Java que actúa como motor de Servlets y Java Server Pages. Desarrollada bajo el proyecto Jakarta de la Fundación Apache (en inglés: Apache Software Foundation). Está desarrollado en un entorno abierto Open Source por lo que su implementación es libre. Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

1.4.3.4 Gestor de base de datos

Una base de datos se le llama a un conjunto de datos informativos organizados en un mismo contexto para su uso y vinculación. Los gestores de bases de datos son un conjunto de programas que permiten almacenar la información, modificarla y extraerla de una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. Es decir se entiende por base de datos un conjunto de datos y como gestor de base de datos una aplicación capaz de manejar este conjunto de datos de manera eficiente y cómoda. Dentro de los gestores que más se emplean se tienen: PostgresSQL, Oracle y Microsoft SQL Server, a continuación se brinda una breve descripción del gestor Microsoft SQL Server 2005 que será usado en la presente investigación.

Microsoft SQL Server 2005 es un sistema para la gestión de bases de datos producido por Microsoft, basado en el modelo relacional. Entre sus principales características se encuentran: la escalabilidad, estabilidad y seguridad, tiene un sistema de reserva de memoria muy rápido, proporciona sistemas de almacenamiento transaccional y no transaccional, permite de forma sencilla añadir otro sistema de almacenamiento y administrar información de otros servidores de datos. Sus lenguajes de consulta son el SQL[6] y el T-SQL Apache (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del servidor. Probado con un amplio rango de compiladores diferentes, escrito en C y en C++. Requiere para su funcionamiento el sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización (25).

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.

[6]Standard Query Language

1.4.4 Tecnologías. Marcos de trabajo

En el desarrollo de software un marco de trabajo (conocido del inglés como framework), es una estructura conceptual y tecnológica de soporte definido con módulos de software concretos, sirviendo de base para que otro proyecto de software pueda ser fácilmente organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y otras herramientas, para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Los marcos de trabajos son básicamente una estructura de soporte que determina la arquitectura sobre la cual puede ser desarrollado un software. Diseñados fundamentalmente para acelerar el proceso de desarrollo, reutilizar código y promover buenas prácticas de desarrollo como el uso de patrones. Existen diversos tipos de marcos de trabajos en la industria del software, con propósitos distintos, algunos están orientados al desarrollo de aplicaciones web. El entorno de desarrollo utilizado J2EE incluye diferentes marcos de trabajos de utilidad tales como Spring, Hibernate, Dojo, Tapestry, WebWork, entre otros.

1.4.4.1 Marcos de trabajo del lado del servidor

Existe un conjunto de marcos de trabajos utilizados en el servidor que hacen posible la aplicación del patrón MVC[7] y que a su vez poseen una elevada aceptación a nivel mundial, ellos son: Struts framework, el cual es libre, permite reducir el tiempo de desarrollo y es compatible con todas las plataformas en las que Java Enterprise esté disponible; JBoss Seam framework, desarrollado por JBoss, donde se puede acceder a cualquier componente EJB (en inglés: Enterprise JavaBeans) desde la capa de presentación refiriéndose a él mediante su nombre de componente y Spring Framework, incluido por la plataforma J2EE. De las tecnologías anteriores se seleccionó Spring Framework para su utilización del lado del servidor, a continuación se presentan sus principales características.

Spring Framework 2.5.6

Spring es un popular marco de trabajo de aplicaciones de código abierto, contiene en conjunto de librerías de clases que brindan una estructura conceptual para el desarrollo de aplicaciones basadas en la plataforma Java/J2EE. Ofrece libertad a los desarrolladores en Java y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto. Su funcionamiento se basa en inversión de control e inyección de dependencia, apoyándose en el API de Java Reflection.

[7]Modelo Vista Controlador.

Spring contiene una gran variedad de funcionalidades organizadas en siete módulos:

- Spring Core: "Núcleo", es la parte fundamental del framework ya que provee toda la funcionalidad de Inyección de Dependencias permitiendo administrar la funcionalidad del contenedor de beans.
- Spring Context: "Contexto", provee de herramientas para acceder a los beans de una manera elegante. El paquete de contexto hereda sus características del paquete de beans y añade soporte para mensajería de texto.
- Spring DAO[8]: provee una capa de abstracción de JDBC que elimina la necesidad de teclear código JDBC tedioso y redundante. Permite que la programación del acceso a datos sea un poco más limpia y entendible. Permite administrar transacciones tanto declarativas como programáticas.
- Spring ORM[9]: provee capas de integración para API de mapeo objeto-relacional, incluyendo Hibernate e iBatis. El mapeo de ORM se puede usar en conjunto con otras características de Spring como la administración de transacciones.
- Spring AOP: provee una implementación de programación orientada a aspectos. Sus funcionalidades permiten desacoplar el código de una manera limpia implementando funcionalidades que por lógica y claridad debería estar separada.
- Spring Web: provee características básicas de integración orientado a la web y se encarga de crear el contexto para las aplicaciones web. Incluye soporte para una variedad de tareas como la subida de archivos y la vinculación de parámetros de las peticiones a objetos del negocio.
- Spring Web MVC: provee una implementación Modelo-Vista-Controlador para las aplicaciones web. La implementación de Spring MVC permite una separación entre código de modelo de dominio y las formas web. Esta implementación facilita una serie de clases controladoras, que manejan la lógica de la navegación e interactúa con la capa de negocio de la aplicación web.

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 J2EE y su uso propicia la aplicación del patrón de arquitectura MVC. Además de las facilidades que brinda fue considerada su uso por la experiencia en proyectos exitosos como SIGEP del cual fue tomada la arquitectura base de Quarxo.

[8] *Data Access Object. Objeto de acceso a datos.*
[9] *Object Relational Mapping. Mapeo objeto-relacional.*

Spring WebFlow 2.0.9

Spring WebFlow es un framework y a su vez un subproyecto de Spring, que permite controlar la navegación de la aplicación Web, guiando al usuario a través de una serie de pasos para completar una transacción de la aplicación. Los flujos Web son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas. En Spring WebFlow un flujo se define mediante un archivo XML en el que se especifican las reglas. Spring WebFlow será usado para los flujos complejos que no puedan ser manejados por los controladores definidos en Spring Web MVC, permitiendo de una manera sencilla definir en archivos XML de configuración el desarrollo total del flujo. Su uso se debe a la fácil integración con la plataforma de Spring Web MVC para la definición de un lenguaje declarativo de flujos.

1.4.4.2 Marcos de trabajo del lado del cliente

En las interfaces de usuario se utilizan marcos de trabajos debido a la importancia que estas poseen en los sistemas informáticos, teniendo en cuenta su interacción con los usuarios. Existen diversas librerías desarrolladas para JavaScript que enriquecen las interfaces de usuario y brindan un ambiente de trabajo más limpio para el uso de Ajax[10], entre ellas se destacan: Qooxdoo, de código abierto para el desarrollo de aplicaciones web interactivas que le permite al desarrollador abstraerse del HTML, CSS[11] y DOM[12]; ExtJS, creado con el fin de desarrollar aplicaciones web interactivas usando tecnologías AJAX y DOM, puede ejecutarse como una aplicación independiente y Dojo Toolkit, el cual permite realizar peticiones asincrónicas al servidor e incorpora soporte para el trabajo con la técnica AJAX. De las tecnologías anteriores se seleccionó Dojo Toolkit, a continuación se presentan sus principales características.

Dojo Toolkit 1.3

Dojo Toolkit contiene APIs[13] y controles (en inglés: widgets) que ayudan al desarrollo de aplicaciones web enriquecidas en el cliente, empaquetando los efectos visuales de la interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX. Destacado por permitir el desarrollo de aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, ofrece soporte

[10] Asynchronous JavaScript and XML.
[11] Cascading Style Sheets.
[12] Document Object Model.
[13] Application Programming Interface

para la internacionalización e incluye un gran cúmulo de componentes visuales basados en HTML, CSS y Javascript para enriquecer la interfaz de usuario.

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.

1.4.4.3 Marcos de trabajo para el acceso a datos

Los marcos de trabajo ORM se utilizan para el trabajo con el acceso a datos de una aplicación, entre ellos se puede mencionar a: IBATIS, una solución híbrida con las mejores ideas de las tecnologías ORM y Mapeador de datos (del inglés: Data Mapper), soporta procedimientos almacenados, SQL dinámico, SQL en línea y el Mapeo Objeto Relacional; Spring JDBC, es un módulo perteneciente al framework Spring, el cual posee algunas librerías de clases para trabajar con base de datos a través del API de Java JDBC y brinda soporte para invocar funciones almacenadas y procedimientos e Hibernate Framework, una solución para el lenguaje de programación Java, concebido bajo la filosofía del código abierto. De las tecnologías anteriores se seleccionó Hibernate Framework para su uso en el acceso a datos, a continuación se describen sus principales características.

Hibernate 3.5

Hibernate es un marco que abstrae el trabajo con la base de datos, soportando la manipulación de los datos persistentes objetualmente, a través de ficheros que mapean clases java contra tablas. 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. Soporta la conexión a una gran variedad de servidores de base de datos, tales como: PostgreSQL, Oracle, SQLServer 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. Entre sus funciones se encuentran: permitir transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, persistencia transitiva. Ofrece también un lenguaje de consulta denominado HQL (en inglés: Hibernate Query Language), siendo este una poderosa vía de comunicación entre el programador y la base de datos, debido a que permite realizar consultas basándose en los objetos del negocio y no en las tablas

de la base de datos propiamente dicho, aunque permite la ejecución de consultas SQL (en inglés: Standard Query Language).

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.

1.4.5 Conclusiones Parciales

Luego del desarrollo del capítulo 1 se concluye lo siguiente:

- Producto al estudio realizado a sistemas que se utilizan para informatizar el proceso de autorizo de desembolso, se decidió ubicar la personalización de los autorizos de desembolsos dentro del subsistema Créditos del sistema Quarxo.
- El estudio de la metodología, los lenguajes, las herramientas y las tecnologías definidas en el proyecto Quarxo que serán usadas en el desarrollo del módulo Autorizo de desembolsos, confirmó la validez del uso de las mismas para lograr los objetivos propuestos. Dentro de ellas se encuentran: el Modelo de Desarrollo de Software v1.1 del centro CEIGE; UML como lenguaje de modelado; Visual Paradigm como herramienta CASE para el modelado; Java (en su especificación JEE) como lenguaje de programación, aprovechando las bondades de un conjunto de framework; Eclipse Galileo 3.5, como IDE de desarrollo; Microsoft SQLServer 2005, como gestor de base de datos y Apache Tomcat como servidor de aplicaciones web.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

2.1 Introducción

En el siguiente capítulo se exponen los artefactos generados por las disciplinas Modelado del negocio, Requisitos y Análisis y diseño del modelo de desarrollo de software propuesto por el centro CEIGE. Además se realiza una breve descripción de los elementos fundamentales de la arquitectura utilizada en el desarrollo. En la disciplina Análisis y diseño se construye la estructura de la solución mediante el modelo de diseño donde se obtendrán una serie de artefactos que contribuirán a una visión detallada sobre cada uno de los requerimientos. De igual forma se ejemplifican los patrones utilizados en el diseño de la solución.

2.2 Modelado de negocio

Es la fase del modelo de desarrollo de software destinada a entender la estructura y dinámica de la organización. Se comprende cómo funciona el negocio que se desea automatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Su objetivo fundamental radica en asegurar que los clientes, usuarios finales y desarrolladores tienen un entendimiento común de la organización.

Roger S. Pressman plantea que se debe desarrollar un modelo de negocio y derivar los requisitos del sistema a partir de este. Esto garantiza que el software que se desarrolle responda a las necesidades y a las condiciones de la organización.

2.2.1. Modelo de procesos de negocio

Un proceso de negocio es un conjunto estructurado de actividades, diseñado para producir una salida determinada o lograr un objetivo específico. Cada proceso de negocio tiene sus entradas, funciones y salidas. Las entradas son requisitos que deben tenerse antes de que una función pueda ser aplicada. Cuando una función es aplicada a las entradas de un método, tendremos ciertas salidas resultantes. Los procesos describen cómo es realizado el trabajo en la organización y se caracterizan por ser observables, medibles, mejorables y repetitivos. Los procesos de negocio son la base para comprender mejor la forma en que opera un negocio en sus diferentes áreas y son una herramienta fundamental para acceder a modelos de calidad (ISO 9001-2000 u otros) y eficiencia.

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.

Con el objetivo de comprender los procesos de negocio e identificar las necesidades reales de los clientes en el BNC se realizaron varias entrevistas. La misma es una técnica utilizada para recopilar la información necesaria y analizar las características de la institución generalmente son preguntas realizadas por los analistas de forma verbal a los usuarios del sistema o aquellos que proporcionan algún dato, no es una interrogación sino una forma de dialogar y llegar a un consenso sobre cómo se realizan los procesos en la organización. A continuación se enumeran los procesos de negocios definidos en el BNC para la gestión de los procesos de autorizo de desembolso:

- Pagar autorizo de desembolso.
- Registrar autorizo de desembolso.
- Modificar autorizo de desembolso.

Al realizar el análisis de dichos procesos se generaron una serie de artefactos según la metodología utilizada tales como: modelo conceptual, descripción de proceso de negocio y reglas del negocio. Dichos artefactos fueron validados posteriormente por los funcionales entrevistados.

2.2.1.1. Descripción del proceso

Para apoyar el estudio del proceso Registrar autorizo de desembolso se le realizó al operador de negociaciones del Banco Nacional de Cuba varias entrevistas, todas de manera informal. Las entrevistas realizadas propiciaron el conocimiento necesario sobre los procesos de autorizo de desembolso.
A continuación se muestra la descripción del proceso de negocio Registrar autorizo de desembolso (*ver Tabla 1*).

Objetivo	Registrar autorizo de desembolso
Evento(s) que lo genera(n)	N/A
Pre condiciones	N/A
Marco legal	

	N/A
Clientes internos	Registrar Garantía.
Clientes externos	N/A
Entradas	
Flujo de eventos	
Flujo básico Autorizo de desembolso	
1	Buscar garantía: El operador de negociaciones busca la garantía relacionada con el desembolso que se desea autorizar en el sistema Quarxo.
2	Extraer garantía: El operador de negociaciones extrae por la Transacción General las contingencias de la garantía relacionadas con el desembolso que se desea autorizar.
3	Contabilizar autorizo: El operador de negociaciones inserta por la Transacción General las nuevas contingencias del desembolso.
Pos-condiciones	
1	El desembolso queda autorizado en el sistema.
Salidas	

1. Transacción Contable del AD (I).

Asuntos pendientes

Posibles mejoras al proceso:

- Con el módulo Autorizo de desembolso se pretende que en el autorizo del desembolso se registre la entidad Autorizo de desembolso con un grupo de información estadística importante para el BNC.
- La extracción e inserción de las nuevas contingencias se hará de manera automática. Generalmente la acción de extracción e inserción de contingencias de la garantía se hace porque en este momento se conocen nuevos respaldos de la misma. El autorizo representa la reinserción de las contingencias de la garantía pero ya en montos atendiendo a los respaldos. Una vez que lleguen los vencimientos de la parte financiada entonces se extraen las contingencias y se insertan los vencimientos en cuentas *reales*.

Requisitos funcionales candidatos:

- Autorizar desembolso.
- Actualizar autorizo de desembolso.
- Consultar autorizo de desembolso.
- Buscar autorizo de desembolso.
- Eliminar autorizo de desembolso.

Sobre el proceso:

Se debe comprobar que los importes separados por respaldos sumen el porciento de la parte financiada, que es la parte que nos interesa tratar en cuentas contingentes porque el resto que es la

parte vista se paga directamente similar a en una negociación vista. Por ello solo extraemos e insertamos contingencias de la parte financiada.

Tabla 1. Descripción del proceso registrar factura de autorizo de desembolso.

2.2.1.2. Diagrama del proceso de negocio

Los diagramas de procesos son representaciones gráficas que se obtienen de cualquier tipo de proceso paso a paso. Básicamente describen los pasos que se siguen en toda una secuencia de actividades, dentro de un proceso o un procedimiento. Actualmente en el sistema Quarxo no se da la opción de realizar una correcta gestión de los procesos de autorizo de desembolso pues se realizan de forma manual consultando los datos de la aplicación lo que resulta un proceso engorroso. El diagrama de proceso (ver Figura 2.1) muestra lo descrito anteriormente.

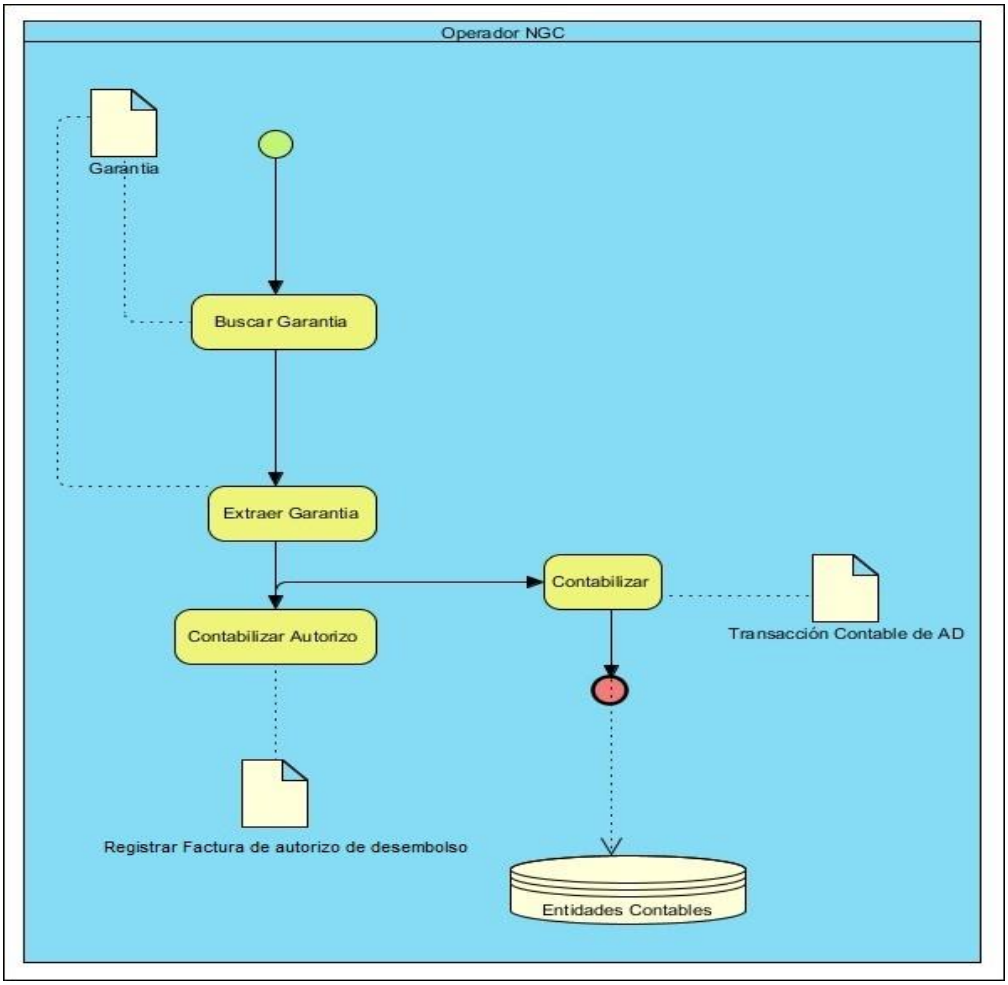

Figura 2.1. Diagrama del proceso Autorizo de Desembolso.

2.2.2 Modelo conceptual

Un modelo conceptual también conocido como modelo de dominio es una representación visual de las clases conceptuales del dominio del problema, explica (a sus creadores) los conceptos significativos indicando las características del proceso en la organización y cómo se relacionan los conceptos más relevantes en la descripción del problema.

El modelo conceptual se describe mediante diagramas UML, específicamente mediante diagramas de clases conceptuales significativas relacionadas con el dominio del problema, donde se trata de obtener el esquema conceptual de la base de datos a partir de la lista descriptiva de objetos y asociaciones identificadas. Su

principal objetivo es establecer un vocabulario común para una mejor comprensión de las clases más importantes, dentro del contexto en el que se realice y facilitar el posterior levantamiento de requisitos.

Para la elaboración del modelo conceptual relacionado al proceso autorizo de desembolso se identificaron las clases conceptuales del negocio, los atributos y las relaciones existentes entre dichas clases. A continuación se muestra el modelo conceptual (ver Figura 2.2) con las clases identificadas y los atributos correspondientes.

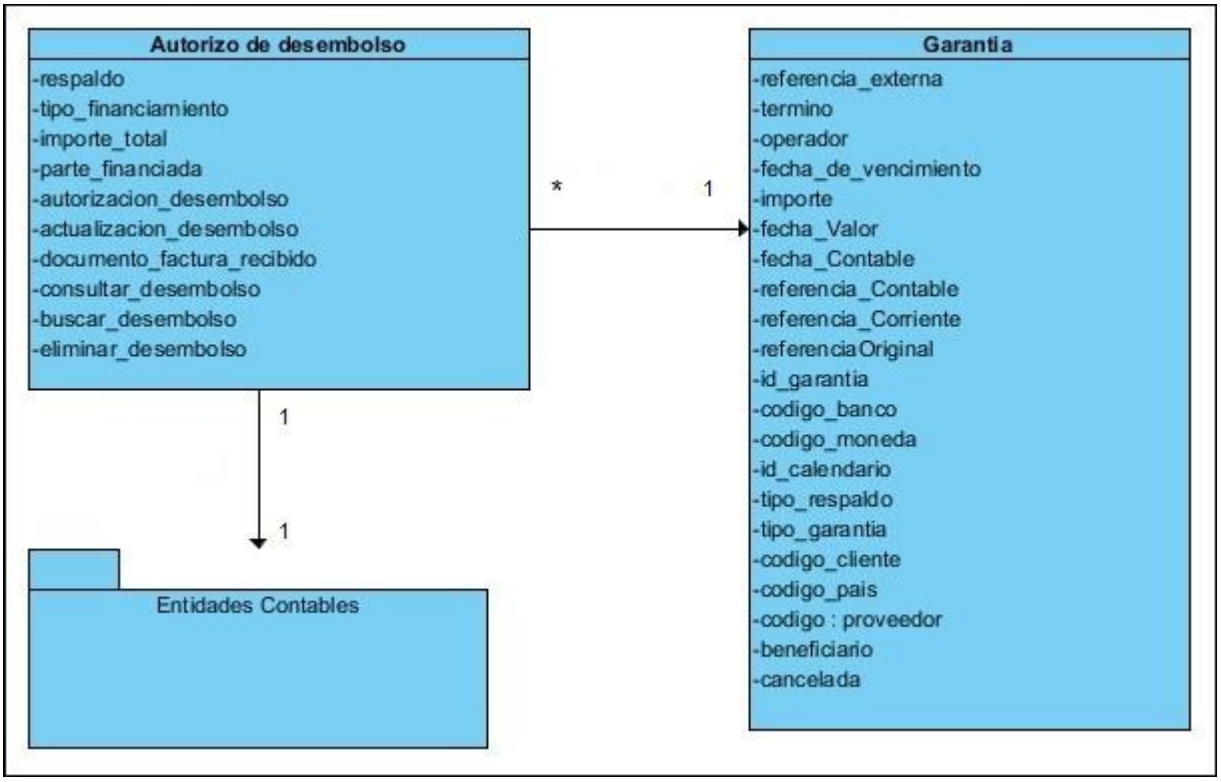

Figura 2.2. Modelo conceptual.

2.2.3 Reglas del negocio

Las reglas son esenciales para los modelos de negocio, son restricciones de comportamiento y/o proporcionan soporte para la dirección de las actividades de negocio se aplican a lo largo de los procesos y procedimientos. Una regla de negocio define o limita un aspecto del negocio con el objetivo de establecer una estructura. Las mismas deben ser explícitas y escritas en un lenguaje natural, expresadas en términos sencillos 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 de negocios se podrán consultar en el expediente del proyecto Quarxo.

2.3 Requisitos

Los requisitos son una condición o capacidad que debe cumplir o poseer un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto o puede ser también una restricción en el proceso de desarrollo de un sistema. La definición de requisitos es una actividad fundamental para modelar el sistema que consiste en identificar qué es lo que realmente se necesita. Esto se realiza con el objetivo de establecer la comunicación entre el cliente y el equipo de desarrollo.

Los requisitos de software son evaluados en dos aspectos fundamentales en cuanto a la criticidad y complejidad seguidamente se detallan los valores.

Interpretación de los valores en cuanto a criticidad:

- Los requisitos con criticidad alta tienen una mayor prioridad a la hora de su implementación para llevar a cabo el desarrollo del componente, ya que sin ellos no se puede dar respuesta a las necesidades imprescindibles del sistema.

- Los requisitos de criticidad media siguen a la hora de su implementación en el desarrollo del componente detrás de los requisitos con criticidad alta, ya que muchos de ellos dan respuestas o son complementos de los mencionados anteriormente.

- Los requisitos de criticidad baja a la hora de su implementación en el desarrollo del componente siguen detrás de los requisitos de criticidad media puesto que muchos de ellos son complementos de los mismos.

Interpretación de los valores en cuanto a complejidad:

- Los requisitos con complejidad alta son aquellos que en su implementación para el desarrollo del componente deben realizar un alto número funcionalidades.

- Los requisitos con complejidad media son aquellos que en su implementación para el desarrollo del componente deben realizar una menor cantidad de funcionalidades.

- Los requisitos con complejidad baja son aquellos que en su implementación para el desarrollo del componente deben realizar un bajo número de funcionalidades.

Los requisitos se pueden clasificar en funcionales y no funcionales. Los primeros son capacidades o condiciones que el sistema debe cumplir y los segundos son propiedades o cualidades que el producto debe poseer. A continuación se muestran los artefactos generados en la disciplina de requisitos del modelo de desarrollo de software propuesto por CEIGE.

2.3.1 Requisitos funcionales

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar. Estos describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Los mismos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema.

Requisitos funcionales (RF): Módulo Autorizo de Desembolso

RF 01. Registrar factura de autorizo de desembolso.

El sistema le permitirá al operador de negociaciones registrar un autorizo de desembolso a cierta garantía registrada.
Prioridad: alta.
Complejidad: media.

RF 02. Eliminar factura de autorizo de desembolso.
El operador de negociaciones tendrá la opción de eliminar una factura de autorizo de desembolso no utilizable para el sistema.
Prioridad: baja.
Complejidad: baja.

RF 03. Pagar autorizo de desembolso.
El sistema permitirá realizar el pago del autorizo de desembolso de una garantía extraída previamente para realizar.
Prioridad: media.
Complejidad: alta.

RF 04. Consultar factura de autorizo de desembolso.
A cualquier autorizo realizado en el sistema se le podrá realizar una consulta para chequear datos.
Prioridad: media.
Complejidad: media.

RF 05. Negociar autorizo de desembolso.
Luego de realizado el registro de una factura, esta podrá ser extraída para negociar acuerdos del mismo, ajustes en el calendario de vencimientos y renegociar el pago de la factura.
Prioridad: alta.
Complejidad: alta.

RF 06. Buscar autorizo de desembolso.

Luego de introducir los criterios de búsqueda el sistema mostrará todos los autorizos de desembolsos realizados hasta el momento.
Prioridad: alta.
Complejidad: media.

RF 07. Actualizar autorizo de desembolso.
Se actualiza el autorizo de desembolso seleccionado y se pueden editar todos sus datos menos el estado.
Prioridad: media.
Complejidad: media.

De los requisitos anteriores se decidió seleccionar Registrar factura de autorizo de desembolso para exponer cada uno de los artefactos generados en esta fase, los mismos se pueden consultar en Anexos 1.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales son exigencias de cualidades que se le imponen al proyecto. Especifican criterios que pueden usarse para calificar las funcionalidades de un sistema en lugar de sus comportamientos específicos. Definen propiedades o características ya sea del sistema, del proyecto o del servicio de soporte, que no son requeridas junto con la especificación del sistema pero que no se satisface añadiendo código, sino cumpliendo con esta como una restricción del software.
De acuerdo con las características sobre las cuales se desarrolla este subsistema se toman en cuenta los RNF[14] correspondientes al sistema Quarxo (Anexo 13).

2.3.3. Descripción de requisitos funcionales

A continuación se muestra la descripción del requisito funcional Registrar factura de autorizo de desembolso (*ver Tabla 2*).

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción.

[14] Requisitos no funcionales.

Flujo de eventos	
Flujo básico Registrar factura de autorizo de desembolso	
1	El operador de negociaciones busca la garantía relacionada con el desembolso que se desea autorizar en el sistema Quarxo.
2	El operador de negociaciones realiza el autorizo.
3	El usuario solicita aceptar.
4	El sistema valida que los datos introducidos sean correctos y muestra el mensaje “Operación realizada satisfactoriamente.”.
5	El usuario acepta el mensaje.
6	Concluye el requisito.
Pos-condiciones	
1	Se registró el autorizo de desembolso.
Flujos alternativos N/A	
1	El usuario corrige los datos.
2	Volver al paso 3 del flujo básico.
3	Información incompleta.
4	El usuario cancela la acción.
Pos-condiciones	
1	No se registra el autorizo de desembolso.
Validaciones	
1	Se validan los datos según lo establecido en el Modelo conceptual: CIG-QF2-N-CRE - i1302.
Relaciones	**Requisitos Incluidos**
	Extensiones N/A.
Conceptos	N/A
Requisitos especiales	N/A.
Asuntos pendientes	N/A.

Tabla 2. Descripción del requisito funcional registrar factura de autorizo de desembolso.

2.3.4. Prototipos de interfaz de usuario

Los prototipos de interfaz de usuario son modelos presentados por el equipo de desarrollo a los especialistas funcionales para que los mismos conciban la simulación del sistema y validen que la interfaz contempla las

necesidades reales. Además son la base para que los desarrolladores implementen las interfaces de usuario finales. Los prototipos de interfaces de usuario presentados en el trabajo de diploma fueron revisados y aprobados por los especialistas del BNC, lo que constituye una validación para los requisitos funcionales.

El siguiente prototipo (Figura 2.3) muestra la interfaz Registrar Factura de Autorizo de Desembolso del requisito en cuestión.

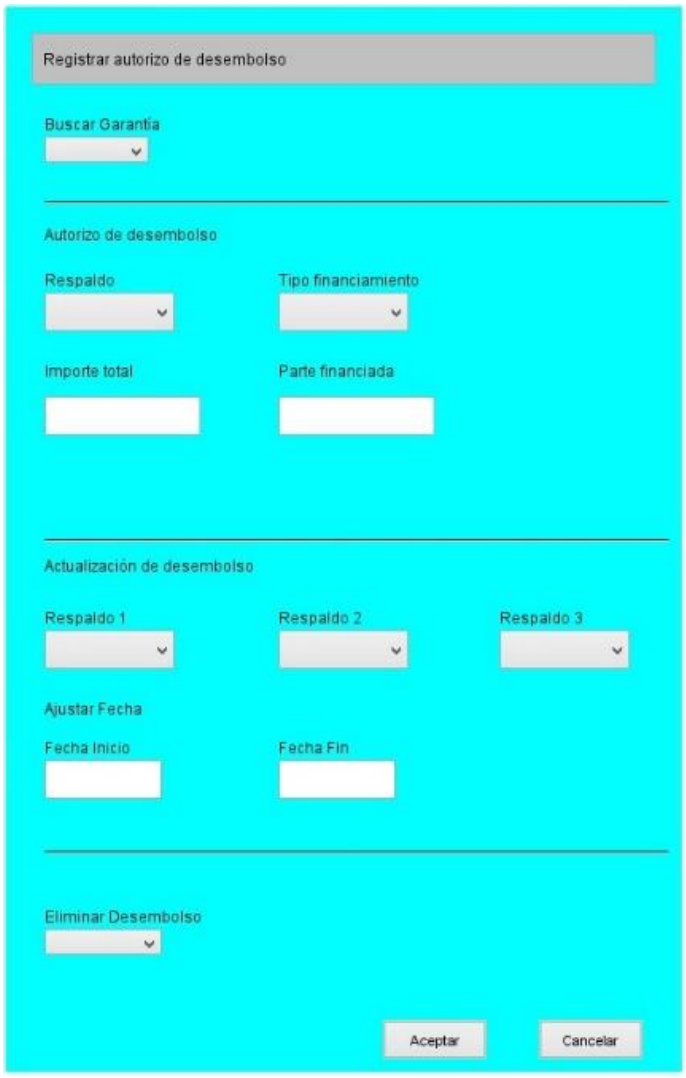

Figura 2.3 Prototipo de interfaz de usuario Registrar factura de autorizo de desembolso

2.3.5. Validación de los requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran descritos correctamente y cumplieran con las necesidades del cliente. Se desarrolló mediante las siguientes técnicas de validación:

- ✓ La revisión técnica por el equipo de analistas principales.

El objetivo de este paso fue verificar la construcción correcta de los artefactos correspondientes en la ingeniería de requisitos.

- La revisión funcional por los especialistas del BNC.

Se realizó con el propósito de validar que las funcionalidades descritas cumplieran con las necesidades y aspiraciones del cliente. En esta revisión se realizó en el BNC con los especialistas funcionales donde se presentaron los prototipos elaborados durante la especificación de requisitos.

- Revisión técnica de artefactos por el equipo de calidad.

Se realizaron revisiones de los artefactos por parte del equipo de calidad del proyecto y se corrigieron las no conformidades identificadas hasta ser liberada la documentación

2.4 Análisis y diseño

El análisis y diseño es una de las etapas de construcción de un sistema informático, que consiste en relevar la información actual, proponer los rasgos generales de la solución futura y realizar el modelado del sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase. Su propósito radica fundamentalmente en generar los artefactos que permiten estructurar el sistema y establecer las bases para la implementación del software.

2.4.1. Modelo de datos

Un modelo de datos es una colección de conceptos bien definidos que ayudan a expresar las propiedades de una aplicación con un uso de datos intensivo. Sirve para describir la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos.

Es necesario resaltar que el subsistema Crédito forma parte del sistema Quarxo el cual ya posee una base de datos. De dicho subsistema se mostrará el diagrama de modelo de datos perteneciente al módulo de Autorizo Desembolso (Figura 2.4).

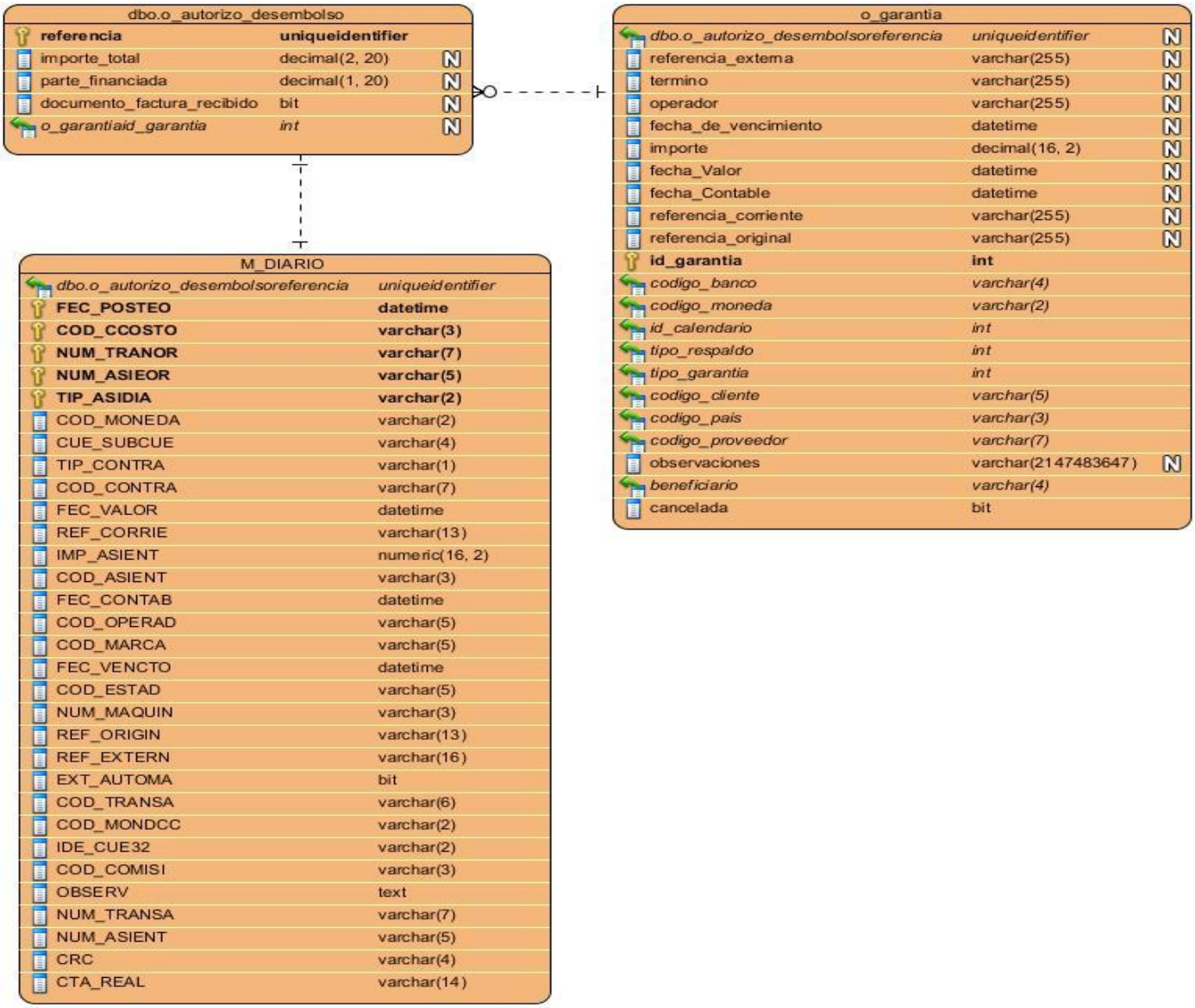

Fig. 2.4 Modelo de datos del módulo Autorizo de desembolso.

2.4.2. Arquitectura del sistema

La arquitectura definida para el desarrollo del sistema Quarxo está basada en una arquitectura de tres capas lógicas fundamentales: capa de presentación, capa de lógica de negocio, capa de acceso a datos y una capa transversal a las otras con los objetos del dominio (Figura 2.5).

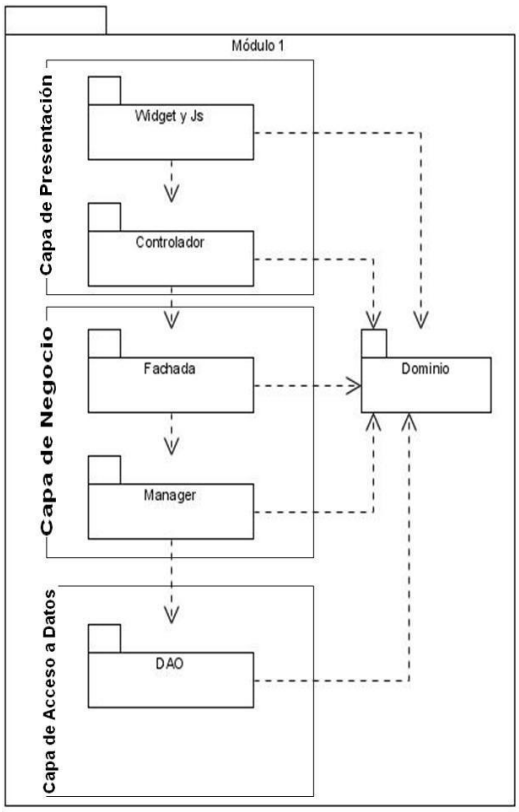

Figura 2.5. Estructura de las capas lógicas del sistema.

2.4.2.1 Capa de presentación

En esta capa se desarrollará la lógica de presentación, es la encargada de interactuar con el usuario, obtiene todos los pedidos de la interfaz, valida y envía los datos al servidor y además, controla la apariencia visual que tendrá la aplicación. En el lado del cliente se utilizará la librería Dojo Toolkit para generar las interfaces que interactuarán con el usuario. En el lado del servidor se utilizará Spring MVC para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente y también se utilizará Spring WebFlow para representar y controlar los flujos complejos reutilizables de la aplicación. La capa de presentación estará relacionada con la capa de negocios y la capa de dominio (Figura 2.6).

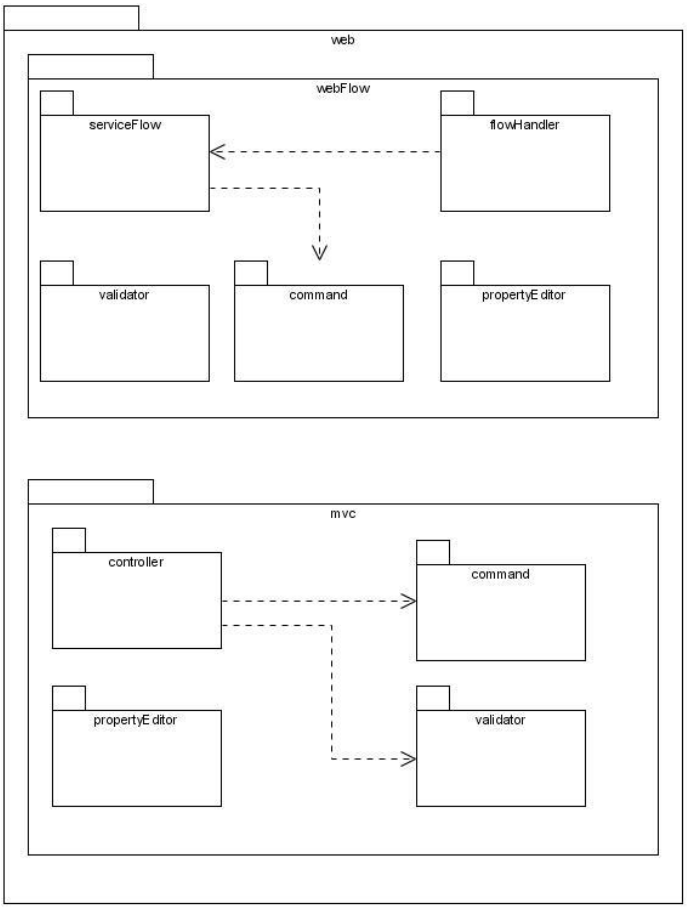

Figura 2.6. Arquitectura de la capa de presentación en el lado del servidor.

2.4.2.2 Capa de lógica de negocio

Esta capa es la encargada de modelar la lógica de negocio, contiene todas las entidades del dominio y los componentes con la responsabilidad de manejar la lógica de negocio de los mismos. Para ello se utiliza el contenedor de Spring Framework, para declarar y representar las relaciones de dependencia de cada una de las clases. Spring Security para asegurar las invocaciones a los métodos. Spring AOP para ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio.

La capa de negocio está divida en dos subcapas: Facade y Manager. La Facade será 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 delegará a la subcapa Manager la realización de la lógica del negocio. Por otro lado, la subcapa Manager tendrá la jerarquía de clases suficiente para implementar el negocio de la aplicación. Esta subcapa utilizará 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 envolverán transaccionalmente las funcionalidades de la aplicación para evitar inconsistencia en los datos. Se utilizará para esto, las políticas de transacciones que propone Spring Framework (Figura 2.7).

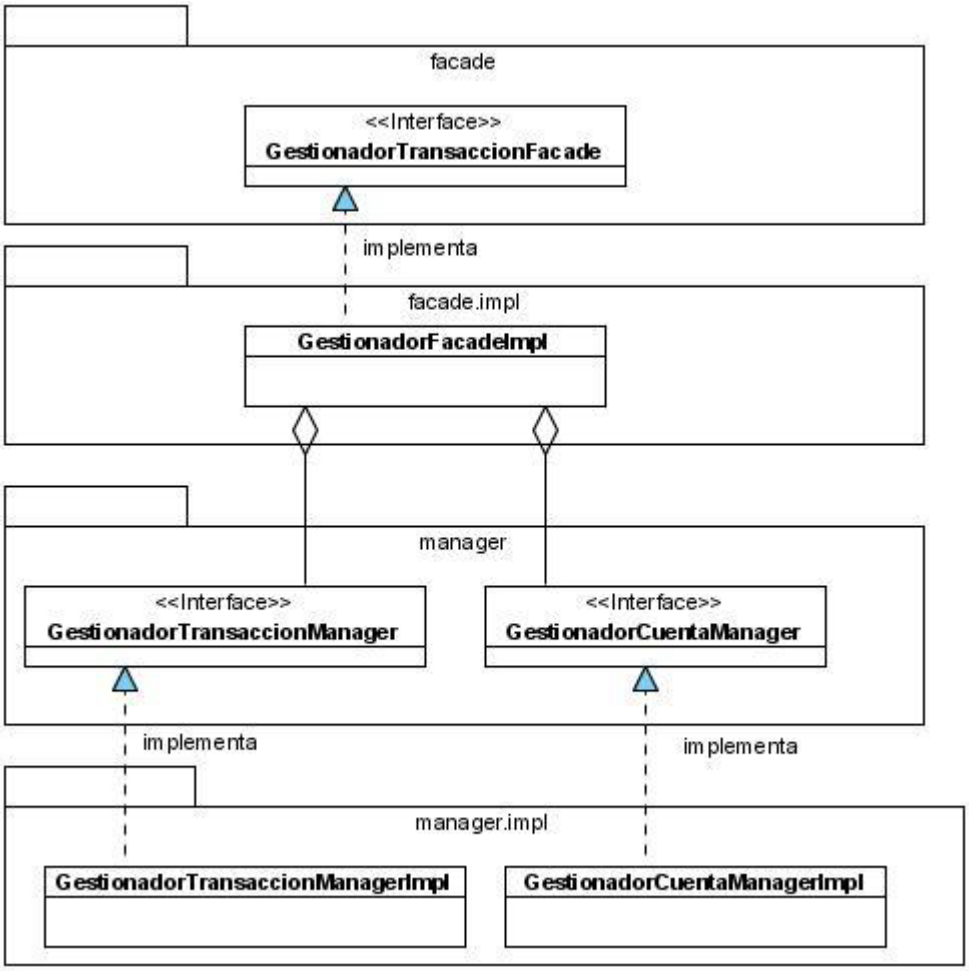

Figura 2.7. Arquitectura de la capa de negocio.

2.4.2.3 Capa de acceso a datos

En esta capa se realizarán todas las operaciones básicas relacionadas con la base de datos tales como: persistir, consultar, actualizar y eliminar los objetos de dominio recibidos de la capa lógica y la invocación a funciones y procedimientos almacenados. La capa de negocio interactuará con esta capa a través de sus interfaces. Para desarrollar esta capa se utilizará el patrón DAO[15]. Las interfaces de los DAOs contienen básicamente las operaciones de inserción, modificación, eliminación y de localización de un objeto a partir de su llave primaria. 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 (ver Figura 2.8) (4)

[15] *Data Access Object*

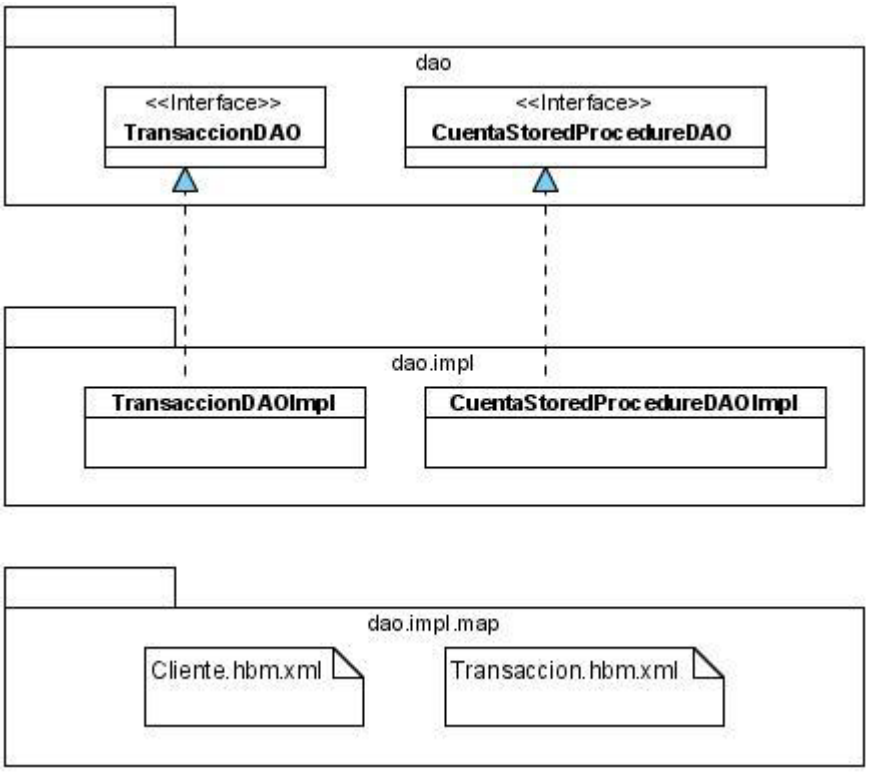

Figura 2.8. Arquitectura de la capa de acceso a datos.

2.4.3. Diagrama de paquetes

Los diagramas de paquetes constituyen una representación jerárquica del sistema o subsistema. Los cuales son utilizados precisamente para mostrar las agrupaciones lógicas en las que se encuentra dividida la aplicación. Estos propician un medio para organizar los artefactos del análisis en fragmentos más fáciles de manejar. Por otro lado ayudan a los desarrolladores mostrando una abstracción más concreta de lo que sería la composición de las diferentes clases en los paquetes.

El siguiente diagrama de paquetes (Figura 2.9) contiene la estructura del módulo de autorizo de desembolso, al cual pertenece el requisito Registrar autorizo de desembolso.

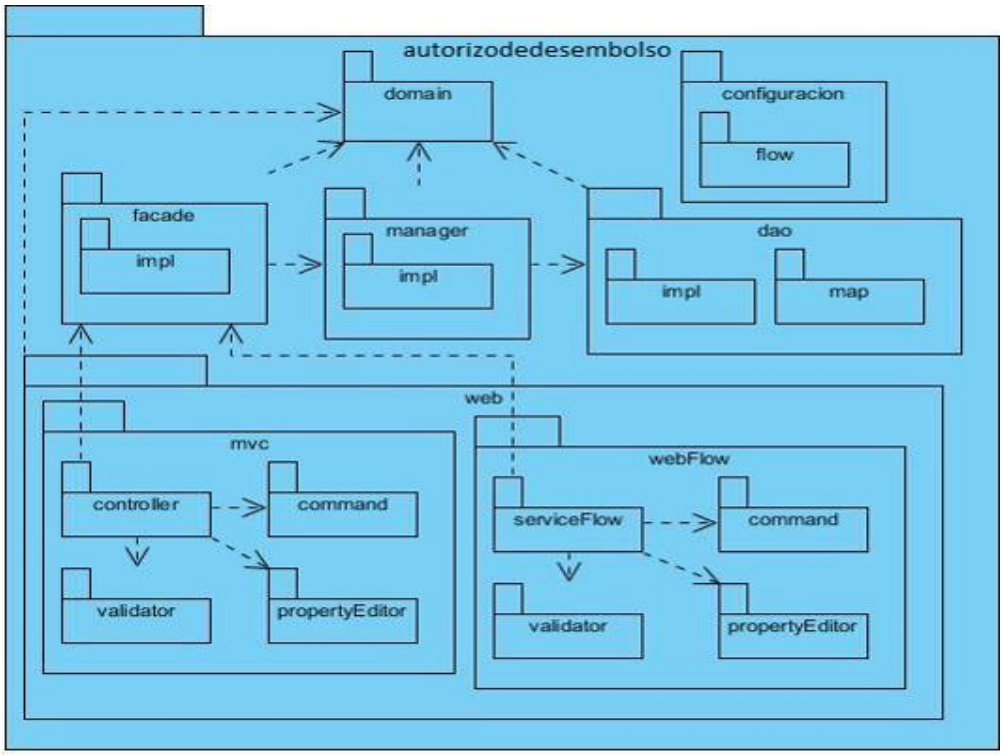

Figura 2.9. Diagrama de paquetes del módulo autorizo de desembolso.

2.4.3.1 Descripción de paquetes

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

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

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.
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 SpringMVC.
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 command: se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.
Paquete propertyEditor: agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa. Paquete validator: cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor.

2.4.4. Diagrama de clases

El diagrama de clases del diseño es un artefacto que permite representar los conceptos en un dominio del problema. Estos diagramas son utilizados durante el proceso de análisis y diseño del software. Describen la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. Constituyen el diseño conceptual de la información que se manejará en el sistema y los componentes que se encargan del funcionamiento y la relación entre uno y otro.
Los siguientes diagramas (ver Figura 2.10, Figura 2.11, Figura 2.12) representan el diseño para la capa de presentación, lógica de negocio y acceso a datos de los módulos negociación y discrepancia atendiendo la arquitectura definida por el proyecto.

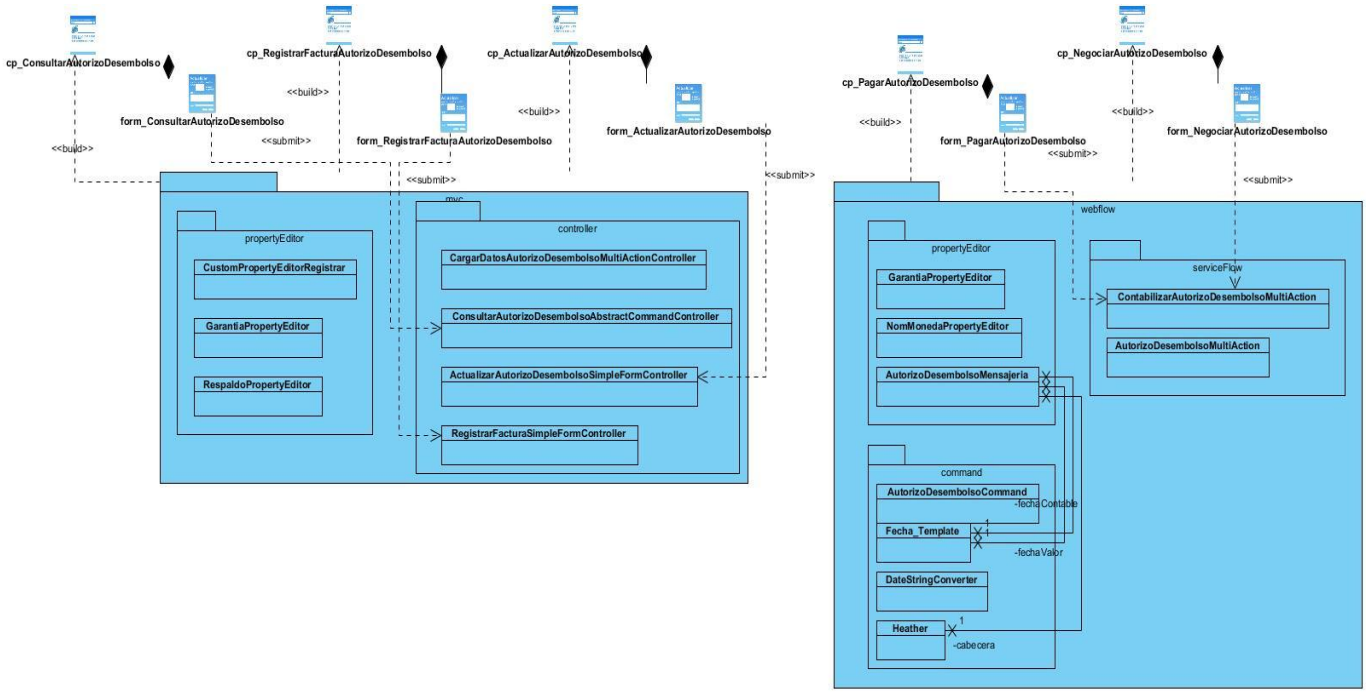

Figura 2.10 Diseño de la capa de presentación del módulo Autorizo de desembolso**.**

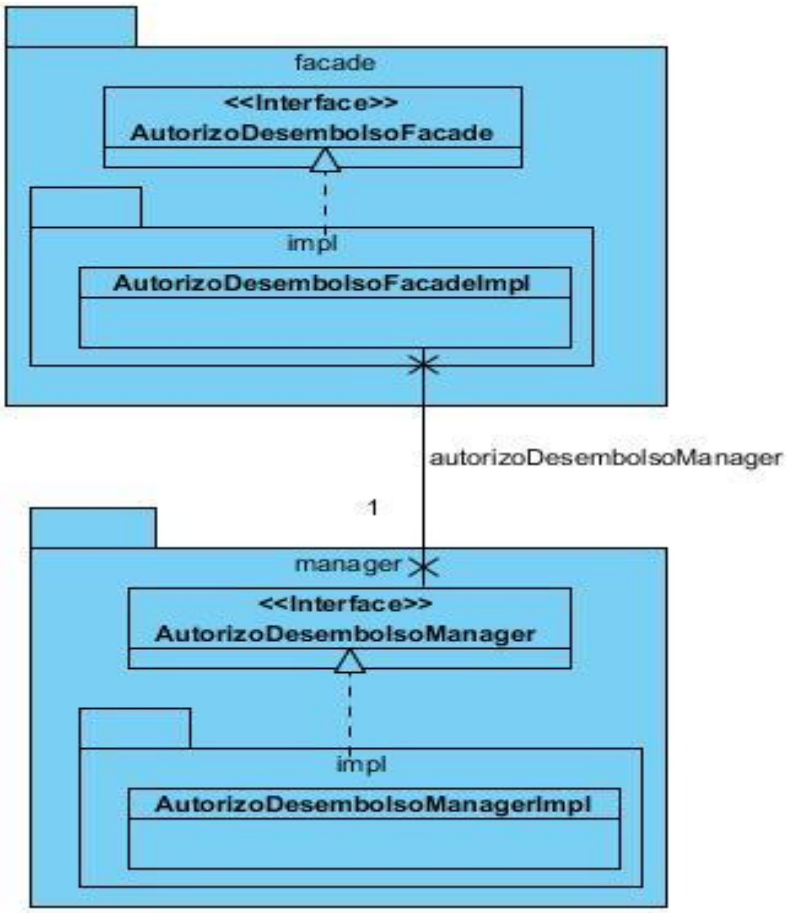

Figura 2.11 Diseño de la capa de negocio del módulo Autorizo de desembolso.

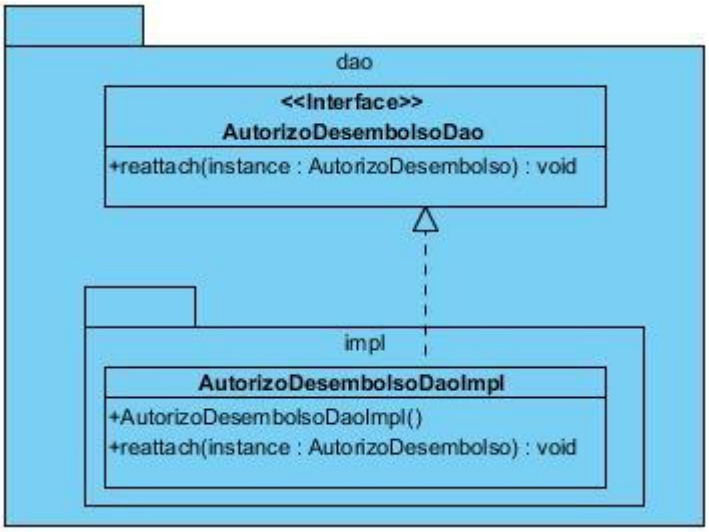

Figura 2.12 Diseño de la capa de acceso a datos del módulo Autorizo de desembolso.

2.4.5 Diagrama de interacción

Los diagramas de interacción son utilizados para modelar los aspectos dinámicos de un sistema. Describen la manera en que interactúan un grupo de objetos entre sí. Existen dos tipos de diagramas de interacción: el diagrama de secuencia y el diagrama de colaboración.

En la etapa del diseño del software se utiliza fundamentalmente el diagrama de secuencia. Este indicará los módulos o clases que forman parte del sistema y las llamadas que se hacen en cada uno de ellos para una tarea determinada. Se realizan los diagramas de secuencia para definir las acciones que se pueden ejecutar y para mostrar las interacciones ordenadas en secuencia entre los objetos de un escenario. Si los requisitos a modelar tienen varios flujos o sub-flujos distintos, suele ser útil crear un diagrama de secuencia para cada uno de ellos. A continuación se muestran los dos diagramas de secuencias (Figura 2.13, Figura 2.14) generados por el requisito registrar autorizo de desembolso.

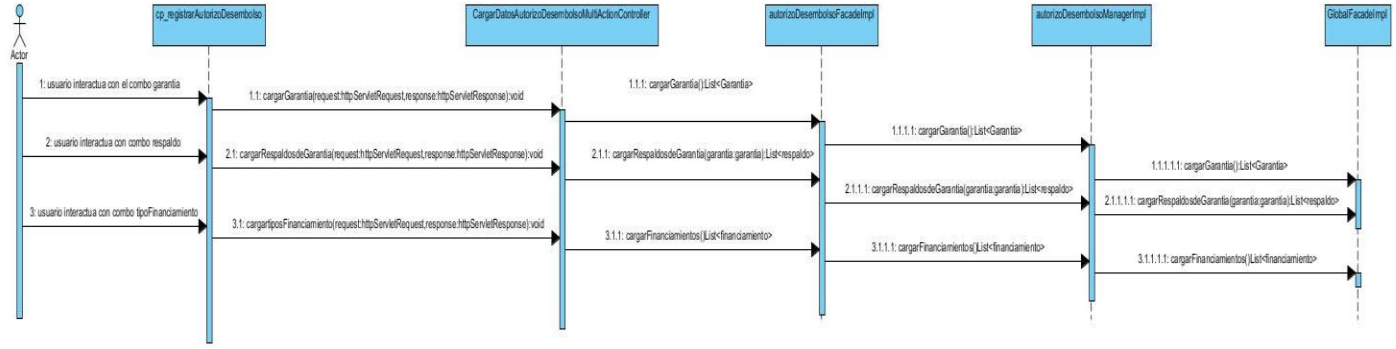

Figura 2.13 Diagrama de secuencia del requisito Registrar factura de autorizo de desembolso primer escenario.

Este diagrama describe la interacción del usuario con la interfaz y el flujo de peticiones para cargar los datos del escenario Registrar Factura de Autorizo de Desembolso, a través de los campos de selección. Comienza cuando Spring MVC construye la página cliente *cp_RegistrarAutorizoDesembolso*. Una vez cargada esta, solicita a *CargarDatosAutorizoDesembolsoMultiActionController* cargar las garantías, la cual le pide a la fachada AutorizoDesembolsoFacadeImpl la lista de garantías la cual se lo solicita al manager AutorizoDesembolsoManagerImpl y esta utiliza directamente *GlobalFacadeImpl que se encarga dhe buscar las garantías*. De igual forma el usuario solicita cargar los respaldos q se le asociaran a la garantía seleccionada a la página cliente cp_RegistrarAutorizoDesembolso la cual solicita a *CargarDatosAutorizoDesembolsoMultiActionController* cargar los respaldos, esta se lo pide a la fachada AutorizoDesembolsoFacadeImpl y esta utiliza al manager AutorizoDesembolsoManagerImpl el cual llama a la fachada GlobalFacadeImpl que se encarga de listar los respaldos. Así concluye este escenario resultando lista la página para continuar con el flujo de sucesos del requisito Registrar Factura de Autorizo de Desembolso.

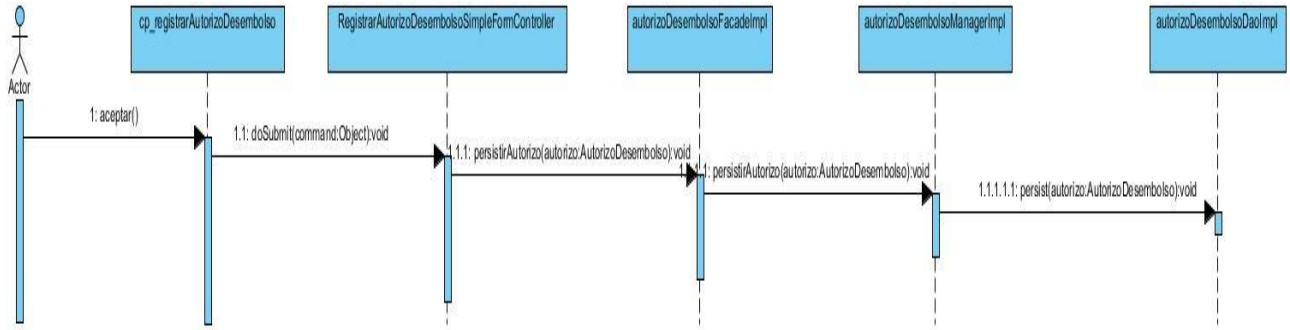

Figura 2.14. Diagrama de secuencia del requisito Registrar factura de autorizo desembolso segundo escenario.

El diagrama describe el flujo de datos partiendo que el usuario ha introducido toda la información en el formulario de la página cliente y ha presionado el botón aceptar. Estos datos son validados por el mismo formulario y enviados al motor de SpringMVC, quien a su vez hace uso de los propertyEditors GarantiaPropertyEditor y RespaldoPropertyEditor para convertir a objetos. El flujo continua con el Controlador RegistrarAutorizoDesembolsoSimpleFormController procesa los datos obtenidos y los envía a la fachada AutorizoDesembolsoFacadeImpl la cual llama al manager AutorizoDesembolsoManagerImpl para que este envíe el autorizo de desembolso a AutorizoDesembolsoDaoImpl el cual se encarga de insertarlo en la base de datos.

2.4.6. Diseño de casos de pruebas

Los casos de pruebas son un conjunto de condiciones o variables bajo las cuales se determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Se pueden realizar muchos casos de prueba para determinar que un requisito es completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de prueba para cada requisito.

Lo que caracteriza un escrito formal de caso de prueba es que hay una entrada conocida y una salida esperada, los cuales son formulados antes de que se ejecute la prueba. La entrada conocida debe probar una precondición y la salida esperada debe probar la pos-condición (Anexo 13).

2.4.7. Patrones de diseño

En el capítulo anterior se realizó un estudio detallado sobre los patrones de diseño, puesto que son una solución estándar para un problema común de programación resuelto con anterioridad. Uno de los pasos a tener en cuenta cuando se decide desarrollar un proyecto de software es identificar qué patrones pueden ser utilizados. A continuación se describe el uso de los patrones aplicados en el diseño de los requisitos a incorporar en el módulo Autorizo de desembolso.

De los patrones GRASP, encargado de describir los principios fundamentales de la asignación de responsabilidades, se emplearon específicamente:

- Experto: se evidencia en la asignación de las funcionalidades a las clases de acuerdo con las información que manejan cada una, ejemplo de la utilización del patrón se encuentra en la clase *AutorizoDesembolsoDAO* responsable de efectuar las operaciones persistir, listar, consultar y eliminar las facturas de autorizo de desembolso realizadas a cada garantía.
- Controlador: un ejemplo de la utilización de este patrón lo representa la clase *CargarDatosAutorizoDesembolsoMultiActionController* debido a que funciona como intermediaria entre la interfaz de usuario y el negocio, presenta la responsabilidad de contener los métodos que cargan los datos que serán mostrados al usuario.
- Alta Cohesión: los paquetes están estructurados con la menor cantidad posible de clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.
- Bajo Acoplamiento: este patrón se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz *AutorizoDesembolsoFacade* y su implementación, ya que la clase de la presentación *RegistrarAutorizoDesembolsoSimpleFormController solo* se relaciona con la interfaz para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

De los patrones estructurales GoF, encargado de describir como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades, se utilizó el patrón:

- Fachada: se evidencia en la utilización de las interfaces en los distintos requisitos ejemplo *AutorizoDesembolsoFacade* responsable 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, reduciendo las dependencias en el sistema entre la parte del servidor y la parte del cliente.

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

Patrón de acceso a datos utilizado:

- DAO: se evidencia con la definición de la clase *AutorizoDesembolsoDAO* efectuando su uso a través del framework Hibernate para almacenar los datos de la discrepancia.

Patrón de presentación utilizado:

Vistas Compuestas: se evidencia en el fichero *pagarAutorizoDesembolso.jsp* que contiene código HTML para construir la estructura de la interfaz mostrada al usuario y para dar cumplimiento a la funcionalidad hace uso de la etiqueta "include" para contener otra vista en su estructura.

Patrón de arquitectura empleado:

- El patrón MVC se evidencia en el módulo Autorizo de Desembolso de forma general, pues utiliza a SpringMVC como framework núcleo de la aplicación, lo cual obliga al mismo a utilizar el patrón MVC para su correcto funcionamiento. Además forma parte de la arquitectura utilizada en el proyecto SAGEB dividida en tres capas: presentación, lógica de negocio y acceso a datos.

2.5 Validación del diseño

El diseño está enfocado a convertir los requisitos del cliente en un modelo que al ser implementado, se obtenga el producto deseado. Generalmente constituye el punto de partida para el desarrollo de software una vez especificados los requerimientos del sistema (32). Evidentemente, realizar una validación del mismo para verificar su calidad y flexibilidad, garantiza una buena base para la implementación. Con este objetivo se utilizan un conjunto de métricas de software orientadas a determinar, qué características del modelo de diseño se pueden estimar para comprobar que el sistema será fácil de implementar en cuanto a organización. Se seleccionaron las métricas **Tamaño Operacional de Clase y Relaciones entre Clases** para validar el diseño del módulo Autorizo de Desembolso

Total de clases	**19**
Total de procedimientos	171
Total de relaciones	34

Tabla 3.Total de clases, procedimientos y relaciones.

2.5.1 Tamaño Operacional de Clase (TOC)

Esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación. Por otro lado, en cuanto menor sea el valor medio para el tamaño más probable es que las clases tengan menos responsabilidad y complejidad y más nivel de reutilización.

Para evaluar con las métricas es necesario definir los valores de los umbrales para los parámetros de calidad (*ver Tabla 4).* Algunos especialistas plantean umbrales para esta métrica basándose en el promedio de operaciones por clases obtenidos, estos valores fueron los aplicados en el diseño del sistema.

Métrica Tamaño operacional de clase			
Atributo	**Categoría**	**Criterio**	**Cantidad de Clases**
Responsabilidad	Baja	< =Prom.	13
	Media	Entre Prom. y 2* Pom.	3
	Alta	> 2* Prom.	3
Complejidad implementación	Baja	< =Prom.	13
	Media	Entre Prom. y 2* Pom.	3
	Alta	> 2* Prom.	3
Reutilización	Baja	> 2*Prom.	3
	Media	Entre Prom. y 2* Pom.	3
	Alta	<= Prom.	13

Tabla 4. Umbrales de medición para TOC.

A continuación se representan los estados de los atributos de calidad:

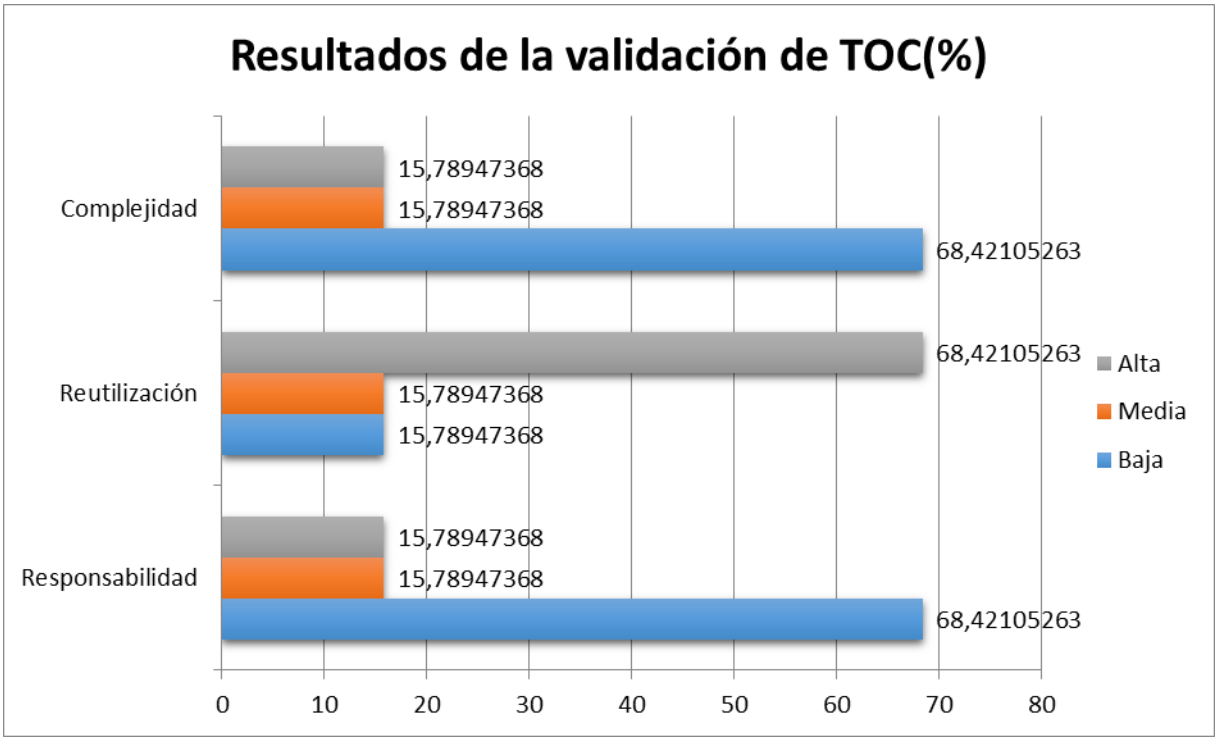

Figura 2.15 Resultados de los atributos de calidad para la métrica TOC

La aplicación de la métrica TOC demuestra que las clases no se encuentran muy sobrecargadas en responsabilidad y presentan un alto nivel de reutilización.

2.5.2 Relaciones entre Clases (RC)

Esta métrica se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases, y es inversamente proporcional al grado de reutilización de las mismas.

Métrica relaciones entre clases(RC)			
Atributo	**Categoría**	**Criterio**	**Total de Clases**
Acoplamiento	Ninguna	0	3
	Baja	1	8

	Media	2	5
	Alta	> 2	3
Complejidad de mantenimiento	Baja	<= PO	11
	Media	Entre PO y 2 * PO	5
	Alta	> 2 * PO	3
Reutilización	Baja	> 2 * PO	3
	Media	Entre PO y 2 * PO	5
	Alta	<= PO	11
Cantidad de pruebas	Baja	<= PO	11
	Media	Entre PO y 2 * PO	5
	Alta	> 2 * PO	3

Tabla 5. Umbrales de medición para RC

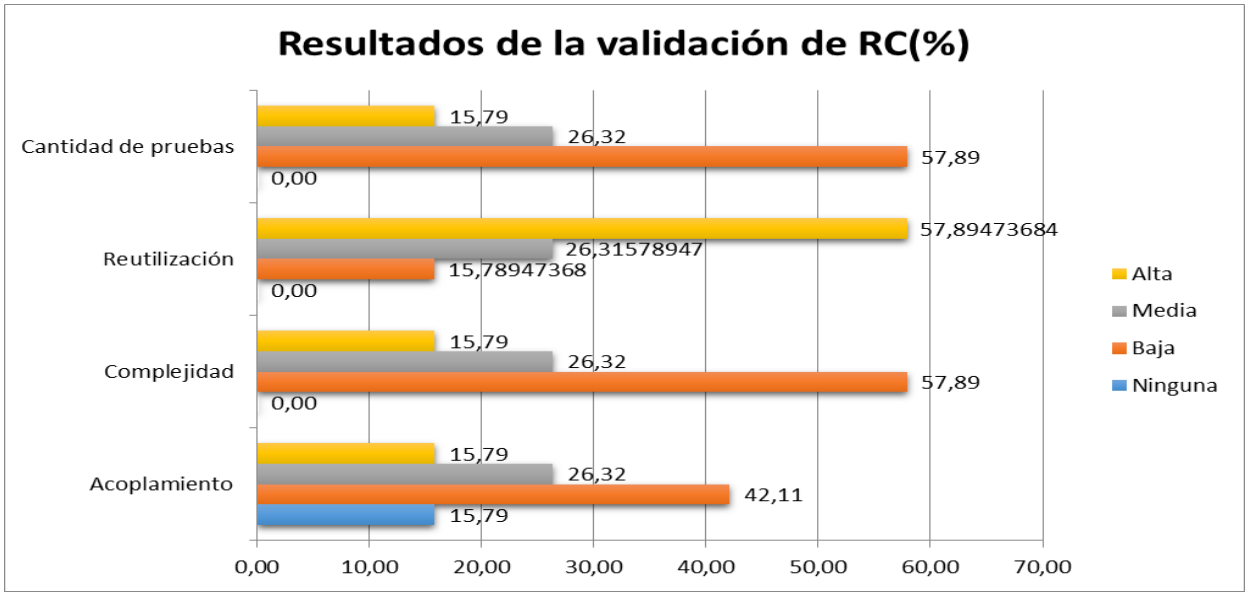

Figura 2.16 Resultados de los atributos de calidad para la métrica RC.

Concluido el análisis de los resultados obtenidos acerca de la evaluación de la métrica RC se puede deducir que existe además bajo acoplamiento entre las clases y presentan un alto nivel de reutilización. Indican además que el diseño no es complejo, pues la complejidad de mantenimiento es baja, así como la complejidad en las pruebas.

2.6 Conclusiones del capítulo

En este capítulo se llevó a cabo las tres primeras disciplinas del ciclo de vida del proyecto Quarxo atendiendo a la metodología defina en el capítulo anterior. Las mismas arrojaron los siguientes resultados:

- La realización del modelado de negocio permitió obtener una visión detallada de los procesos de negocio de la entidad. Mediante las entrevistas efectuadas a los especialistas del BNC se obtuvieron los artefactos correspondientes a esta disciplina y como efecto se logró la identificación y descripción de los procesos de negocios, la confección de los diagramas de procesos utilizando la notación BPMN, el modelo de dominio usando como lenguaje de modelado UML y la descripción de las reglas de negocio de cada uno de los procesos identificados.
- La disciplina de requisitos posibilitó la identificación y descripción de todos los requisitos funcionales a incorporar en la solución. Logrando el entendimiento claro por parte del equipo de desarrollo de las necesidades del cliente. La validación de los requisitos se realizó a partir de la técnica Prototipos, la cual permitió validar que las funcionalidades descritas cumplían con las aspiraciones del cliente.
- Como parte de la disciplina Análisis y diseño se utilizaron las métricas TOC y RC para la validación del diseño propuesto las cuales arrojaron resultados satisfactorios para los atributos de calidad evaluados.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

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 Modelo de implementación

El modelo de implementación describe la forma en que los elementos del modelo del diseño y las clases se implementan en términos de componentes, como ficheros de código fuente o ejecutables. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación los lenguajes de programación utilizados y cómo dependen los componentes unos de otros (19).

3.2.1 Diagrama de componentes

Los diagramas de componentes modelan la vista estática de un sistema. Se representan como un grafo de componentes de software unidos por medio de relaciones de dependencia (compilación, ejecución). Los mismos permiten visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe un apartado del sistema.

Para el desarrollo del sistema Quarxo se crean un grupo de componentes de los que consumirán servicios los subsistemas y módulos de la aplicación. Se muestra a continuación el diagrama (ver Figura 3.1) que evidencia la interacción del subsistema Crédito con los componentes definidos y una descripción de la función que realiza cada uno de ellos.

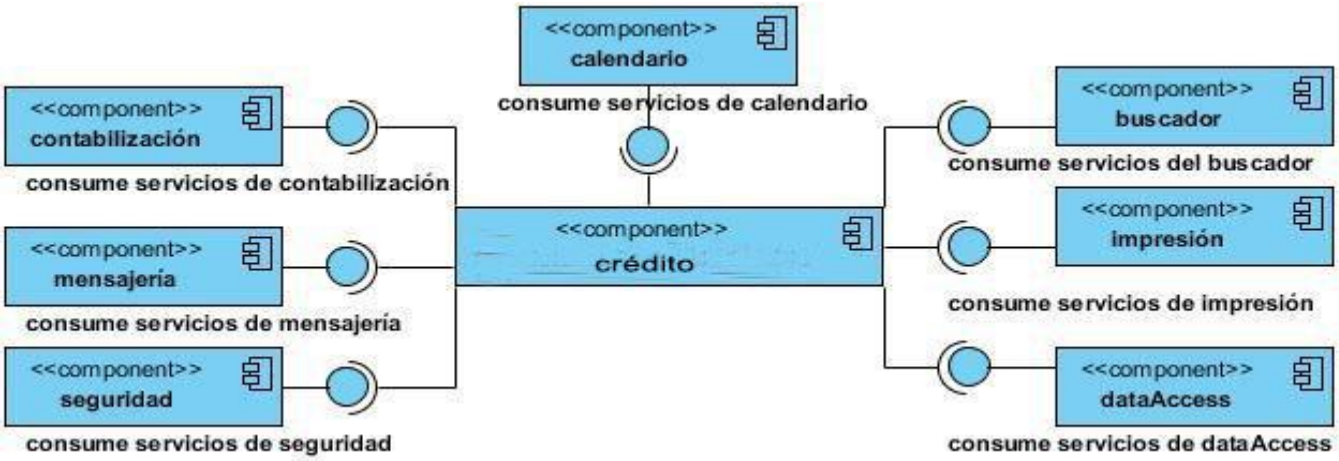

Figura 3.1 Diagrama de componentes del subsistema Crédito.

El componente **mensajería** brinda un conjunto de clases y funcionalidades que garantizan el envío de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos, la recepción y el envío automático de los mensajes resultantes de la contabilidad, disminuyéndose el esfuerzo de los usuarios del sistema para enviar o recibir dichos mensajes. Su utilización se evidencia en la en el pago y negociación de un autorizo de desembolso.

El componente **seguridad** se encarga de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y permisos. Gestiona la información de los usuarios, roles y sus permisos para lograr la seguridad requerida en el sistema. Es utilizado en el subsistema para controlar el acceso del usuario a las posibles operaciones a realizar sobre los autorizos de desembolsos.

El componente **contabilización** brinda un conjunto de clases y funcionalidades necesarias que permiten realizar la contabilización de determinadas operaciones; constituye el de mayor consumo por parte del subsistema.

El componente **dataAccess** es imprescindible para el funcionamiento de todo el sistema, porque se encarga de proporcionar los elementos necesarios para la conexión a la base de datos.

El componente **buscador** brinda un conjunto de clases que constituyen el motor de búsqueda, así como una interfaz gráfica encargada de la búsqueda de diferentes conceptos.

El componente **impresión** contiene las funcionalidades necesarias mediante las cuales es posible la impresión de la información referente a la contabilidad.

El componente **calendario** ofrece un conjunto de clases además de una interfaz gráfica encargadas de gestionar los elementos necesarios para la creación de los cronogramas de compromisos de pago por cada funcionalidad para la gestión de las formas de pago que contienen los vencimientos.

3.3 Estándares de codificación

Un estándar de codificación comprende todos los aspectos relacionados con la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. Estos no están enfocados a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código (16). En general, los estándares de codificación son reglas que se siguen para la escritura del código fuente con el objetivo facilitar la compresión y permitir que los desarrolladores del equipo de proyecto puedan relacionar cada porción del mismo como si se tratara de su propio código garantizando además que el trabajo se realice de forma organizada.

3.3.1 Convenciones de nomenclatura

El proyecto SAGEB 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. Ejemplo: *AutorizoDesembolsoManager*. En este caso el nombre de clase está compuesto por dos palabras iniciadas cada una con letra mayúscula. Para la nomenclatura de las variables y los métodos se definió la notación CamelCasing, que es muy similar a PascalCasing con la excepción de que la letra inicial siempre es minúscula, en caso de estar compuesta por más de una palabra, estas empezarán con mayúscula exceptuando la primera. Ejemplo: *idAutorizoDesembolso*. Lo mismo se aplica a los nombres de ficheros de código javascript y sus funciones y variables internas. Ejemplo: *registrarAutorizoDesembolso***.** Los nombres

de los paquetes deben ser con minúscula y deben ser sustantivos que describan de alguna forma que tienen dentro. Ejemplo: *autorizodesembolso*, *web*, *dao*.

3.3.2 Convenciones en la capa de presentación

Las clases pertenecientes a los sub-paquetes mvc y webflow del paquete web se nombrarán de la siguiente manera:

- Paquete controller: [Nombre de la clase] + [Nombre del controlador de Spring que se hereda]. Ejemplo:*CargarDatosAutorizoDesembolsoMultiActionController.*
- Paquete command: [Nombre de la clase] + [Command]. Ejemplo: *AutorizoDesembolsoCommand*.
- Paquete propertyEditor: [Nombre de la clase] + [PropertyEditor].Ejemplo: *GarantiaPropertyEditor.*
- Paquete serviceFlow: [Nombre de la clase] + [Action o de MultiAction en dependencia de cuál de las dos clases herede]. Ejemplo: ContabilizarAutorizoDesembolsoMultiAction.

3.3.3 Convenciones en la capa de negocio

Las clases pertenecientes a los paquetes facade y manager y sus sub-paquetes impl respectivamente se nombrarán de la manera siguiente:

✓ Paquete facade: para las Interfaces [Nombre del módulo] + [Facade]. Ejemplo: *AutorizoDesembolsoFacade* y las clases que implementan las interfaces [Nombre del módulo] + [FacadeImpl]. Ejemplo: *AutorizoDesembolsoFacadeImpl.*

✓ Paquete manager: para las Interfaces [Nombre del módulo] + [Manager]. Ejemplo: *AutorizoDesembolsoManager* y las clases que implementan las interfaces Nombre del módulo] + [ManagerImpl]. Ejemplo: *AutorizoDesembolsoManagerImpl*.

3.3.4 Convenciones en la capa de acceso a datos

Las clases pertenecientes al paquete dao y su sub-paquete impl se nombrarán de la manera siguiente:

✓ Paquete dao: para las Interfaces [Nombre de la entidad] + [DAO]. Ejemplo: *AutorizoDesembolsoDAO* y las clases que implementan las interfaces [Nombre de la entidad] + [DAOImpl]. Ejemplo: *AutorizoDesembolsoDAOImpl.*

3.4 Validación de la solución

Durante la implementación del software las posibilidades de cometer errores son múltiples por lo que se hace necesario realizar pruebas que contribuyan a detectar las imperfecciones e irregularidades de la aplicación.

Las pruebas realizadas al módulo Autorizo de desembolso son las correspondientes a los niveles Unidad e Integración.

3.4.1 Pruebas unitarias

Las pruebas unitarias 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. Las pruebas unitarias se derivan en pruebas funcionales (caja negra) y pruebas estructurales (caja blanca). En la presente investigación se llevaron a cabo como parte de las pruebas unitarias, las pruebas de Caja Blanca a través de la técnica Camino Básico y de forma automatizada mediante el framework JUnit.

3.4.1.1 Pruebas estructurales o caja blanca

El método de caja blanca es aplicado con el objetivo de analizar la lógica interna del programa. En la presente solución se determinan los caminos posibles dentro de los métodos de las clases a través de la técnica Complejidad Ciclomática. Luego, mediante el marco de trabajo JUnit, el cual permite la automatización de las pruebas de aplicaciones en Java integrándolo al IDE de desarrollo Eclipse, se aplicó la técnica Camino Básico.

A continuación se muestran las imágenes que corresponden a la realización de las pruebas de Caja Blanca aplicando el framework JUnit al método cargarGarantias *(ver figura 3.2)* de la clase controladora CargarDatosAutorizoDesembolsoMultiActionController.

```
public void cargarGarantias(HttpServletRequest request,
        HttpServletResponse response) {
    JSONObject json = new JSONObject();
    JSONArray array = new JSONArray();
    List<Garantia> garantias =autorizoDesembolsoFacade.listarGarantias();
    JSONObject data = new JSONObject();
    for (int i = 0; i < garantias.size(); i++) {
        data = new JSONObject();
        Garantia garantia = garantias.get(i);
        data.put("value", garantias.get(i).getIdGarantia());
        data.put("innerHTML", garantias.get(i).getReferenciaOriginal());
        array.add(data);
    }

    json.put("identifier", "value");
    json.put("label", "innerHTML");
    json.put("value", "value");
    json.put("items", array);
    try {
        ResponseUtil.escribirDatosEnElResponse(response, json);
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Figura 3.2 Método cargarGarantias de la clase AutorizoDesembolsoMultiActionController

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). Lo anterior 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.

```
public class CargarDatosAutorizoDesembolsoMultiActionTestCase extends TestCase {

    CargarDatosAutorizoDesembolsoMultiActionController controlador;
    private MockRequestControlContext controlHttpRequest;
    private HttpServletRequest mockHttpServletRequest;
    private MockRequestControlContext controlHttpResponse;
    private HttpServletResponse mockHttpServletResponse;
    private MockRequestControlContext controlHttpSession;
    private HttpSession mockHttpSession;

        @Before
    public void setUp() throws Exception {
        controlador = new CargarDatosAutorizoDesembolsoMultiActionController();
        ApplicationContext context= new ClassPathXmlApplicationContext("classpath:cu/uci/finixubnc/junit/autorizodesembolso-context.xml");
        AutorizoDesembolsoManagerImpl manager=(AutorizoDesembolsoManagerImpl)context.getBean("autorizoDesembolsoManager");
        AutorizoDesembolsoFacadeImpl facade= new AutorizoDesembolsoFacadeImpl();
        facade.setAutorizoDesembolsoManager(manager);
        controlador.setAutorizoDesembolsoFacade(facade);

        mockHttpServletRequest=(HttpServletRequest)controlHttpRequest.getMockExternalContext();
        mockHttpServletResponse=(HttpServletResponse)controlHttpResponse.getMockExternalContext();
        mockHttpSession= (HttpSession)controlHttpSession.getMockExternalContext();
        super.setUp();
    }
```

Figura 3.3 Creación del contexto en la clase TestJUnit 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 *autorizodesembolso-context.xml,* el cual va a ser el encargado de declarar y mapear los objetos creados por cada nivel de la aplicación.

```
    @After
    public void tearDown() throws Exception {
        controlHttpRequest.getClass();
    }
    @Test
    public void testCargarGarantia() throws Exception{
        mockHttpServletRequest.getParameter("idAutorizoDesembolso");

        boolean cargar=controlador.cargarGarantias(mockHttpServletRequest, mockHttpServletResponse);
        assertEquals(true, controlador.cargarGarantias(mockHttpServletRequest, mockHttpServletResponse));


    }
```

Figura 3.4 Método tearDown y testCargarGarantia de la clase TestJUnit.

En el método *tearDown* es donde se verifican si las llamadas a los métodos se realizaron correctamente, es el encargado de informar la existencia de los errores en la realización de las pruebas. En este caso verifica los objetos que son pasados por el *request* de forma automática para todos los test definidos en la clase *TestJUnit. (ver figura 3.4).*

Durante la realización de la prueba al método, se preparan los parámetros de pruebas necesarios para el método *cargarGarantias.* Dichos parámetros serán pasados por el *mockHttpServletRequest*. Posteriormente se realiza la condición de prueba a través del método *assertEquals* el cual compara un valor 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 caso favorable para dicha prueba.

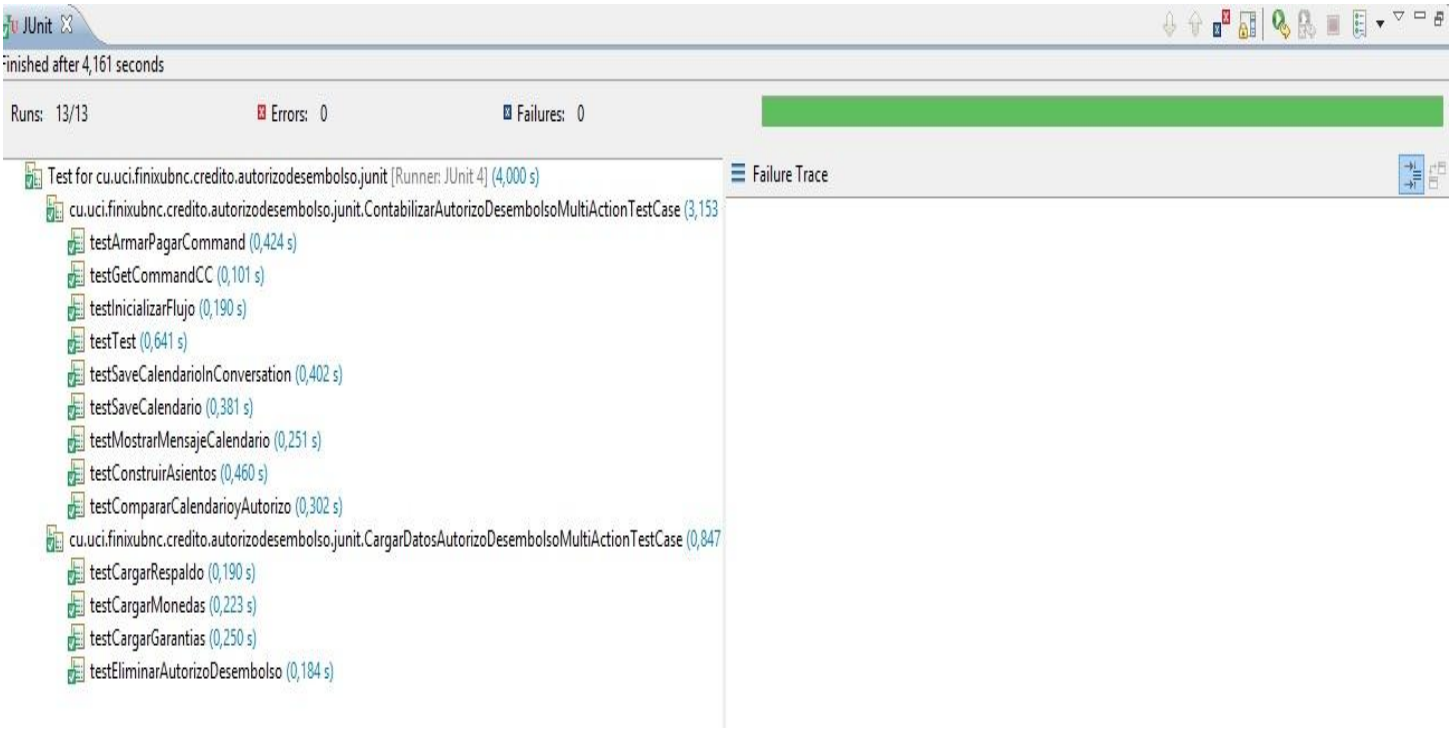

Figura 3.5 Consola de JUnit con el resultado de la prueba realizada.

3.4.1.2 Pruebas funcionales o caja negra

Se realizó por parte de un grupo de profesionales del proyecto la revisión del módulo a partir de los documentos de casos de pruebas de cada requisito. Las no conformidades encontradas en las dos revisiones realizadas se muestran a continuación:

Cantidad de No conformidades	**Iteración 1**	**Iteración 2**

	6	2

Tabla 6. Pruebas del módulo Autorizo de desembolso.

3.4.2 Validación de la variable de investigación

La investigación desarrollada plantea como idea a defender que con el desarrollo del módulo Autorizo de desembolso para el sistema Quarxo se permitirá la gestión personalizada del proceso, la cual contribuirá a mejorar el proceso en el BNC. La mejoría fue evaluada teniendo en cuenta las variables tiempo y cantidad de información entrada por los usuarios. A continuación se evaluará la variable tiempo en las operaciones de autorizo de desembolso, la variable cantidad de información no necesita ser evaluada por cuanto significa que el proceso se haya personalizado.

Tiempo de ejecución de los procesos en la gestión de autorizo de desembolso.

Operación	**Tiempo en minutos en el módulo Autorizo de desembolso sin informatización.**	**Tiempo en minutos en el módulo Autorizo de desembolso informatizado.**
Registrar Autorizo de desembolso	10	2

Tabla 1. Valoración de la variable tiempo en el módulo Autorizo de desembolso antes y después de la informatización.

3.5. Conclusiones parciales

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

- El modelo de diseño realizado como parte de las disciplinas previas a la implementación, constituyó a pesar de pequeños cambios que debieron hacerse, una guía fiel para el trabajo de los desarrolladores.
- El uso de las metodologías y herramientas definidas permitieron desarrollar con éxito la disciplina Implementación en el módulo Autorizo de Desembolso.

- 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 módulo desarrollado cumple con las expectativas del cliente y podrá ser implantado próximamente en el BNC.

CONCLUSIONES

Una vez culminado el presente trabajo de diploma se puede afirmar que se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas al inicio de la investigación. El desempeño de los objetivos específicos permitió arribar a las siguientes conclusiones:

- El estudio y caracterización de las tecnologías y herramientas definidas para el desarrollo de la aplicación contribuyeron a su correcta utilización garantizando la disminución de costos por concepto de licencias y soporte.
- Las disciplinas Modelado del negocio y Requisitos del modelo de desarrollo propuesto por CEIGE contribuyeron a una mayor comprensión del negocio de la entidad y entendimiento de las necesidades del cliente; permitiendo realizar la captura, especificación y validación de los requisitos.
- Los artefactos generados durante el desarrollo de la disciplina Análisis y diseño, proporcionaron la base para la disciplina Implementación, incidiendo notablemente en la estructura, organización y reutilización de código a través del diagrama de clases del diseño y de los patrones de diseños empleados.
- La disciplina Implementación proporcionó la creación del módulo Autorizo de desembolso y la integración satisfactoria del mismo al sistema Quarxo.
- Con el desarrollo del presente trabajo se consolidaron los conocimientos adquiridos durante la carrera que permitieron la realización del mismo. Se logró una buena interrelación con los clientes del BNC, lo que permitió alcanzar el objetivo general y cumplir las tareas definidas para el desarrollo de la solución.

RECOMENDACIONES

A partir del estudio realizado y las experiencias acumuladas a lo largo del desarrollo de la investigación, el autor propone las siguientes recomendaciones:

- Continuar la investigación del tema a fin de mejorar la solución propuesta incorporando en futuras versiones nuevas funcionalidades, implicando los estudios realizados y las herramientas utilizadas en el presente trabajo de diploma.
- Desarrollar una versión genérica del sistema con el objetivo de ser comercializada y utilizada por otras entidades bancarias en el ámbito internacional.

REFERENCIAS BIBLIOGRÁFICAS

1. **Cobas Portuondo, Jorge Luís, Romeu Valle, Aliuska y Macías Carrasco, Yoel**. La investigación científica como componente del proceso formativo. La Habana: PODIUM, 2010. Vol. II.
2. Economía. Sistemas Bancarios. [En línea] 2012. [Citado el: 15 de marzo de 2014.] http://economiaes.com/bancos/bancarios-sistemas.html.
3. Banco Central de Cuba. Sistema Bancario. [En línea] 2009. [Citado el: 15 de marzo de 2014.] http://www.bc.gov.cu/espanol/sist_bancario.asp.
4. Programación Web. Programación del lado del servidor. [En línea] [Citado el: 16 de abril de 2014.] http://www.prograweb.com.mx/pweb/0203ladoServidor.html.
5. **Pérez, Ismaury Figueroa y Ledón, Alain Sánchez**. Análisis y diseño del Módulo emisión de Carta de Crédito. 2009.
6. Banco Central de Cuba. Bancos Comerciales. [En línea] 2009. [Citado el: 17 de marzo de 2014.] http://www.bc.gov.cu/espanol/bancos_comerciales.asp#BNC.
7. **Castillo, Miguel**. LA CONTABILIDAD GUBERNAMENTAL EN CUBA. 2008.
8. **Medina, Maria de los Angeles. Scribd.** CAMPOS DE LA APLICACIÓN DE LA CONTABILIDAD. [En línea] 2010. [Citado el: 17 de marzo de 2014.] http://es.scribd.com/doc/50595325/CAMPOS-DE-LA-APLICACION-DE-LA-CONTABILIDAD.
9. **Zunino, Gustavo.** Despachante de aduanas paraguay. Instrumentos de pago en el comercio internacional. [En línea] 16 de Mayo de 2009. [Citado el: 19 de marzo de 2014.] http://serviciosaduanerosycomerciales.wordpress.com/?s=Instrumentos+de+pago.
10. Zapata, Cristina I. De Gerencia. [En línea] Documentación de embarque internacional, 2003. [Citado el: 19 de marzode 2014.] ttp://www.degerencia.com/articulo/documentacion_de_embarque_internacional_guia_practica/imp.
11. **UCP500**. Reglas y Usos Uniformes de la CCI para Créditos Documentarios.
12. **Josar, Cristina**. Gerencie. la contabilidad y el sistema contable. [En línea] 28 de agosto de 2008. [Citado el: 22 de marzo de 2014.] http://www.gerencie.com/sistemas-contables.html.
13. COBIS SCI. *Sistema de comercio internacional.* [En línea] [Citado el: 22 de marzo de 2014.] http://www.oocities.org.htm.

14. Global Trade Managament (GTM). *Compliance Content Software.* [En línea] [Citado el:22 de marzo de 2014.] http://www.questaweb.com.
15. **Rodríguez, Roxana Bermúdez.** *Diseño e implementación de los módulos Documentos de los módulos Documentos de embarque, Discrepancia y Negociación de Quarxo para el Banco Nacional de Cuba.*
16. **Hernández, Liliana María y Toirac, Aramis Stalyn Pérez**. Definición de los Requerimientos Funcionales del Módulo Emisión de Carta de Créditos. 2008.
17. **Jacobson, Ivar, Booch, Grady y Rumbauch, James**. EL Proceso Unificado de Desarrollo de Software. s.l.: Pearson Educación, 2000.
18. CEIGE. Modelo de desarrollo de software v1.1. 2012.
19. **López Rodríguez, Yoan Antonio y Matías León, Yulier**. DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO TESORERÍA, PRÉSTAMOS Y DEPÓSITOS DEL PROYECTO BANCO NACIONAL. 2008.
20. **Larman, Craig**. UML y Patrones. s.l.: Prentice Hall, 1999.
21. Object Management Group. Unified Modeling Lenguage. [En línea] [Citado el: 1 de abril de 2014.] http://www.uml.org.
22. BIZAGI. Bizagi Process Modeler. [En línea] [Citado el: 4 de abril de 2014.] http://www.bizagi.com/esp/descargas/BPMNbyExample.pdf.
23. Babylon. Glosario de Informática. [En línea] 2010. [Citado el: 16 de abril de 2014.] http://diccionario.babylon.com/java/.
24. MySQL. Las principales características de MySQL. [En línea] 2011. [Citado el: 18 de abril de 2014.] http://dev.mysql.com/doc/refman/5.0/es/which-os.html.
25. **López, Yoan Antonio Rodríguez, y otros**. Sistema para la Gestión de las Cartas de Créditos.
26. **Ladd, Seth y Donald, Keith**. Expert Spring MVC and Web Flows. s.l.: Apress, 2006.
27. **A.Russell, Matthew**. Dojo The Definitive Guide. 2008. 34. Peak, Patrick y Heudecker, Nick. Hibernate Quickly. s.l.: Manning Publications.CO, 2006.
28. **Welicki, León**. MSDN. Patrones y Antipatrones. [En línea] 2012. [Citado el: 19 de abril de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.
29. **CAMACHO, ERIKA, CARDESO, FABIO y NUÑEZ, GABRIEL**. ARQUITECTURAS DE SOFTWARE. 2004.
30. **Pressman, Roger S**. Ingenería de Software. Un enfoque práctico. 1998. Vol. I.