**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**Facultad 3**


# Desarrollo del subsistema Vencimiento de Quarxo v2.0

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

Autor: Annier Puig Madrazo

Tutor: Ing. Daylen Barbán Reyes

Ing. Javier Martínez Muñoz

La Habana

2012

*"El conocimiento nos hace responsables."*

*"No se vive celebrando victorias, sino superando derrotas."*

*CHE*

**DECLARACIÓN DE AUTORÍA.**

Declaro que soy el único autor de este trabajo y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmo el presente a los ___ días del mes de junio del año 2012.

**Annier Puig Madrazo**

_____

Firma del Autor

**Ing. Daylen Barban Reyes.**

_____

Firma del Tutor

**Ing. Javier Martínez Muñoz**

_____

Firma del Tutor

**DATOS DE CONTACTO.**

Ing. Javier Martínez Muñoz: Graduado en la Universidad de las Ciencias Informáticas. Instructor. Profesor de Matemática III y IV. 4 años de experiencia en el tema. 3 años de graduados. Analista principal de proyecto SAGEB.

Correo electrónico: jmmunoz@uci.cu

Ing. Daylen Barbán Reyes: Graduada en la Universidad de las Ciencias Informáticas.

Correo electrónico: dlbarban@uci.cu

**AGRADECIMIENTOS.**

*Quiero agradecer a mi familia de manera especial a mi madre por todo el amor apoyo y confianza que ha depositado en mí. A mis abuelos por ser las personas más buenas y cariñosas que existen en este mundo.*

*A mis tutores por todo el apoyo que me han brindado durante este tiempo.*

*A mis amigos que han sabido estar ahí de una forma u otra en los momentos difíciles para darme su apoyo: Jorge Reinaldo, Franky, Reinier, Yasmany y Osmany.*

*A ese gran grupo 3509 que es lo mejor en esta universidad:*

*A mis compañeros de estudio desde el primer año, que de una manera u otra me influyeron en este logro: Vladimir, Eduardo, Evelio, Leonardo, Dariel, Hector, Yakelin y Arlethy.*

*Al amor de mi vida por ser tan comprensiva, dedicada y cariñosa.*

**DEDICATORIA.**

*Este logro se lo dedico a toda mi familia, mis amigos y mis compañeros de estudio y en especial a:*

*Mi madre, por ser la mejor del mundo, darme su cariño y apoyo en todo momento.*

*A mi abuela, mi abuelo y mi bisabuela: por haberme criado y dado todo el amor que necesita una persona cuando nace.*

*A mi tía que aun siendo tan reservada también supo darme su apoyo.*

*Y a una de las personas más importantes en mi vida en estos momentos: mi novia Libet Quiala González por el amor incondicional que me brinda cada día.*

## RESUMEN

El sistema Quarxo desarrollado por el proyecto SAGEB de la Universidad de la Ciencias Informáticas (UCI) cuenta con el subsistema Vencimiento que incluye todas las funcionalidades del proceso de gestión de los vencimientos, que responden a los requisitos pactados en una primera fase de desarrollo del producto. Las operaciones sobre los vencimientos que no son satisfechas por ese subsistema actualmente se realizan por la Transacción General de Contabilidad, constituyendo una vía de solución lenta y compleja en comparación con Vencimiento, exigiendo mayor esfuerzo y revisión por parte del operario, sobre todo cuando involucra mucha contabilidad, actualización de operativos y envío de mensajería bancaria. Por tal motivo se decide desarrollar una segunda versión del subsistema Vencimiento que incluya esas funcionalidades.

El objetivo del presente trabajo es solucionar el problema en el Banco Nacional de Cuba a partir de la Captura de los Requisitos Funcionales, el Análisis, Diseño e Implementación de la solución, así como demostrar a través de las pruebas correspondientes que la segunda fase del subsistema Vencimiento representa una mejor opción para el cliente. Con este objetivo se realiza una caracterización de la metodología, patrones de diseño, herramientas y tecnologías de la plataforma Java utilizadas, así como 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.

**Palabras claves:**

Análisis, Banco Nacional de Cuba, Diseño, Implementación, Requisitos, Vencimiento.

# ÍNDICE

## Índice de Figuras.

## Índice de Tablas.

## INTRODUCCIÓN.

En el mundo actual la utilización de las tecnologías informáticas se ha generalizado, generando incontables beneficios para la sociedad. Las computadoras y las aplicaciones conocidas como software están presentes prácticamente en la mayoría de los lugares donde se maneje información, con fines de automatización, reducción de costos, incremento del control, la eficiencia y la seguridad. La industria bancaria desde algunos años ha venido haciendo uso de los sistemas informáticos para mejorar sus procesos y servicios. El gran volumen y sensibilidad de datos que manejan los bancos, la complejidad y el dinamismo que pueden presentar sus procesos, así como el aumento de la competencia, han provocado la necesidad de que incluso hasta los más conservadores, inviertan en innovación e incorporación de tecnología para hacer que sus sistemas automatizados sean cada vez más rápidos, seguros y eficientes.

En el sistema bancario cubano no existe fuerte presión competitiva, pero la necesidad de modernización de las aplicaciones informáticas en algunos bancos ha aumentado producto a cambios ocurridos en la esfera en los últimos años, así como a transformaciones específicas en sus procesos. El Banco Nacional de Cuba (BNC) es un ejemplo fehaciente de esta realidad. Como parte de la búsqueda de soluciones internas factibles a problemas sociales y económicos, y en aras de satisfacer las actuales necesidades operacionales del BNC con eficiencia, seguridad y mínimo costo, se desarrolla en la Universidad de las Ciencias Informáticas (UCI) por el proyecto SAGEB[1] un sistema informático llamado Quarxo.

Quarxo posee un conjunto de subsistemas que resuelven procesos de negocio específicos que pueden estar relacionados entre sí. Uno de los más críticos del sistema lo constituye la gestión de los vencimientos o compromisos de cobros o pagos, debido a la cantidad y complejidad de las operaciones involucradas. Para el manejo de los vencimientos se encuentra el subsistema Vencimiento, que cubre todos los requisitos pactados en una primera fase del desarrollo de Quarxo.

Existe un conjunto de requisitos que surgieron a raíz de nuevos cambios introducidos en el sector bancario y otros, que por cuestiones de tiempo de desarrollo quedaron fuera del alcance de la primera fase y pendientes a satisfacerse en una segunda versión de Quarxo.

Actualmente los procesos de registrar importaciones y exportaciones, cobrar comisiones de cuenta única, pagar comisiones, condonar, registrar y actualizar vencimientos y actualizar renegociaciones, así como la

[11] Sistema Automatizado de Gestión Bancaria.

rebaja de Exportaciones/Importaciones, se realizan por la Transacción General de Contabilidad (TGC), que permite ejecutar cualquier operación bancaria construyendo de forma manual cada registro o asiento contable involucrado. El uso de la TGC puede ser complejo y lento cuando se requiere contabilizar un proceso de negocio que involucra más de 10 asientos, como es el caso de las funcionalidades mencionadas anteriormente, debido a la necesidad de seleccionar las extracciones y especificar las inserciones y reinserciones con sus contrapartidas correspondientes de forma manual. Esto posibilita la existencia de errores en el cálculo de importes y en la introducción de los datos durante la conformación de los asientos contables, debido a que la transacción solo posee validaciones generales de contabilidad y no propias del proceso de negocio que se esté realizando. Otro inconveniente de la TGC es el hecho de que no permite actualizar las tablas operativas con la información contable de forma automática, haciendo necesario recurrir al módulo (Nomencladores) encargado de esta función y reintroducir los mismos datos. La TGC tampoco está integrada con los subsistemas de mensajería de Quarxo impidiéndose el envío de mensajería SLBTR[2] y/o SWIFT[3] de forma automática. Teniendo en cuenta que las operaciones que representan esos procesos sobre los vencimientos, pueden involucrar gran cantidad de asientos contables, exigir la actualización de operativos y el envío de mensajes SLBTR y SWIFT en algunos casos, sus ejecuciones a través de la TGC pueden ser lentas y complejas, representando nuevos requisitos para el sistema.

De acuerdo al estudio de la problemática planteada se define como **problema a resolver** el siguiente: ¿Cómo agilizar los nuevos procesos dentro de la gestión de los vencimientos en Quarxo?

El **objetivo general**: desarrollar una segunda fase del subsistema Vencimiento de Quarxo para agilizar los nuevos procesos dentro de la gestión de los vencimientos en el Banco Nacional de Cuba.

Para darle solución al problema planteado se define como **objeto de estudio**: el proceso de gestión de los vencimientos en los sistemas informáticos contables.

Como **objetivos específicos** se definen los siguientes:

- ✓ Fundamentar la investigación mediante la elaboración del Marco Teórico.

[2] Sistema de Liquidación Bruta en Tiempo Real.
[3] En español Sistema Internacional de Transacciones Financieras y en inglés System Worldwide Internacional Financial Transactions.

✓ Realizar el análisis, diseño e implementación de las nuevas funcionalidades del subsistema Vencimiento.

✓ Validar el análisis, diseño y la implementación de las nuevas funcionalidades del subsistema Vencimiento.

✓ Validar que el subsistema permite una disminución del tiempo de ejecución y la complejidad de los nuevos procesos dentro de la gestión de los vencimientos.

A partir de la relación existente entre el problema, el objetivo general y los objetivos específicos surge como **campo de acción** la informatización de la gestión de los vencimientos en el BNC.

La **idea a defender** en este trabajo es la siguiente: El desarrollo de la segunda fase del subsistema Vencimiento de Quarxo permitirá agilizar los nuevos procesos dentro de la gestión de los vencimientos en el Banco Nacional de Cuba.

Como **resultado se espera** la obtención de una segunda fase del subsistema Vencimiento de Quarxo que permita agilizar los nuevos procesos dentro de la gestión de los vencimientos en el Banco Nacional de Cuba.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

En este capítulo se presenta la fundamentación teórica del tema mediante; la descripción de algunos sistemas bancarios existentes relacionados con el objeto de estudio, la valoración del estado del arte, y una caracterización de las metodologías, tecnologías, lenguaje de programación y herramientas propuestas para darle solución al problema planteado.

CAPÍTULO 2: REQUISITOS Y ANÁLISIS DE LA SOLUCIÓN.

Contiene las descripciones de los requisitos funcionales, obtenidos mediante la utilización de las técnicas de captura de requisitos: entrevistas y tormentas de ideas. Incluye como uno de los artefactos de salida de Requisitos el diagrama de caso de uso del sistema. También se elabora el análisis de la solución, obteniéndose los diagramas de clases del análisis como una primera aproximación al modelo de diseño.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN.

En este capítulo se elabora el diseño a partir de la especificación de requisitos del subsistema Vencimiento, como base para la implementación. Incluye la construcción de diagramas de clases de

diseño, diagramas de paquetes y diagramas de secuencias para conformar el modelo de diseño, según la Arquitectura Base y los requisitos del sistema, así como la realización del modelo de datos. Además como artefactos de la implementación se muestra el diagrama de integración de componentes y los estándares de codificación y convenciones de nomenclatura.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN.

Contiene las pruebas de Caja Blanca y de Caja Negra realizadas una vez concluida la implementación del subsistema: pruebas sobre las funciones internas de un software y los requisitos funcionales del software respectivamente. Incluye además un conjunto de datos reales como resultado de las comprobaciones realizadas en el entorno de despliegue que demuestran que la solución facilita y agiliza los procesos a los que responden los nuevos requisitos del subsistema, así como un aval del cliente evidenciando su satisfacción.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

## 1.1 Introducción.

En este capítulo se presentan los elementos principales de la fundamentación teórica del trabajo: algunos conceptos generales de importante conocimiento para entender el sistema y conceptos específicos de los procesos, así como una valoración sobre la informatización de los vencimientos a nivel nacional e internacional. También se realiza una caracterización de las tecnologías, metodologías, herramientas y lenguajes de programación que se utilizan para el desarrollo de la solución.

## 1.2 Conceptos fundamentales.

**Banco.**

Los bancos son instituciones de tipo financiera encargados de administrar el dinero que se les deja en custodia por sus clientes y utilizarlo para prestárselo a otros individuos o empresas aplicándoles un interés. En la actualidad los bancos modernos realizan múltiples funciones, en contraste con las que hacían en la antigüedad y posteriormente en el medioevo, épocas en que tenían como funciones principales la custodia del dinero y el cambio. Se debe señalar que el banco moderno tiene que cumplir tres grandes funciones que reflejan: La intermediación del crédito, la intermediación de los pagos y la Administración de los capitales.[1]

**Sistema financiero cubano.**

El sistema financiero cubano se encuentra definido por aquellas instituciones que realizan la actividad financiera, rectoradas por el Ministerio de Finanzas y Precios, que constituye el rector de los ingresos, gastos públicos y presupuestos, y se encarga de negociar la deuda externa y todo lo relacionado con el seguro. En la actualidad el sistema financiero cubano se encuentra integrado por varias instituciones entre las que se encuentran:

- ✓ El Banco Central de Cuba (BCC), cuya misión es emitir la moneda nacional y velar por su estabilidad, proponer e implementar la política monetaria del país y actuar como órgano rector del sistema bancario y financiero, asegurar el normal funcionamiento de los pagos internos y externos, así como ejercer las funciones relativas a la disciplina y supervisión de las instituciones financieras.
- ✓ El Banco Nacional de Cuba (BNC), cuyas principales funciones, atribuciones y deberes son: actuar como funcionario corresponsal de bancos extranjeros; obtener y otorgar créditos; mantener el registro, control, servicio y atención de la deuda externa de Cuba y realizar todo tipo de operaciones y negocios de intermediación financiera. Mantiene en sus registros contables la deuda externa del país, la oficial, la

bancaria y la de proveedores e instrumenta las transacciones que se derivan de la renegociación de la deuda.[2]

**Contabilidad bancaria.**

Se ocupa de la capacitación, la medición y la valoración de todos aquellos elementos financieros que circulen internamente en un banco. Es la actividad de control de la información de todo el dinero que circule en una entidad bancaria con el fin de suministrarles a los gerentes bancarios, las herramientas para que puedan realizar la toma decisiones. [3]

**Asiento contable.**

Los asientos contables reflejan hechos contables, que son aquellos sucesos que hacen variar el balance o las cuentas de resultados, debiendo ser registrados para su posterior cómputo. Todas las variaciones contables han de ser medibles en unidades monetarias pues de lo contrario no se podrían contabilizar.[4]

**Vencimiento.**

Un vencimiento es un asiento contable que representa un compromiso u obligación de pago o cobro que se hace exigible en una fecha futura. Se registra a través de transacciones en un libro contable llamado diario donde permanece pendiente de alguna operación.[4]

**Calendario.**

Representa un acuerdo entre las partes donde se considera la fecha de elaboración y pago de la nómina de un período, englobando varias formas de pago que contendrá un vencimiento.[4]

**Condonar.**

Es el proceso mediante el cual se exime del compromiso de pago de uno o varios vencimientos. [5]

**Capitalizar.**

Es adicionar al capital inicial del período/subperíodo la suma de dinero que en concepto de intereses se han generado en el mismo. Pasando así a conformar el capital inicial del siguiente.[5]

**Préstamo.**

Es una acción mediante la cual (naciones o gobiernos, empresas o individuos) le entrega algo a una persona o institución mediante un contrato, con la obligación de restituirlos entregando además un porcentaje de interés o dando otras ventajas a cambio.[5]

**Renegociación.**

Acuerdo entre bancos y deudores en el cual se negocia un nuevo programa de pagos para saldar la deuda.[6]

**Mora.**

Retraso en el cumplimento de las obligaciones, por lo común la de pagar cantidad líquida y vencida.[5]

**SLBTR.**

Sistema de Liquidación Bruta en Tiempo Real. Permite la realización en tiempo real de transacciones u operaciones de pago entre las diferentes entidades, utilizando medios electrónicos de comunicación y teniendo como centro al BCC para monitorear el flujo de las transferencias interbancarias que se realicen.[7]

**SWIFT.**

Es un sistema computacional a nivel mundial de comunicaciones, que permite a los bancos de distintos países intercambiar información relacionada con las operaciones que le son propias. [8]

## 1.3 Sistemas de Gestión Bancaria internacionales y nacionales.

Como parte de la fundamentación teórica de la investigación se realiza un estudio de los sistemas informáticos bancarios tanto nacionales como internacionales que permiten la gestión de vencimientos, con el propósito de verificar la existencia de una solución factible al problema a resolver. Como resultado fueron identificados los siguientes sistemas:

### 1.3.1 Sistemas internacionales.

**OPENBRAVO.**

Openbravo es un sistema ERP (Enterprise ResourcePlanning), en español: Gestión de Recursos Empresariales) de código abierto que permite a los clientes su control total durante todo el tiempo de uso. Brinda a las organizaciones mejoras en su rendimiento empresarial mediante una mayor productividad y agilidad en el negocio. Posee diferentes módulos entre los que se encuentra el de Gestión económico-financiera, el cual se divide en tres grandes bloques, uno de ellos es el de Gestión de cobros y pagos. Este bloque contiene funciones para la gestión de las cuentas a cobrar y las cuentas a pagar en una empresa, como son:

- ✓ Extracto bancario.
- ✓ Diario de caja.
- ✓ Remesas.
- ✓ Liquidación.

- ✓ Liquidación manual.[9]

Este sistema en general permite realizar los cobros y pagos en una empresa, a través del módulo de Gestión económico-financiera. Las operaciones que se realizan en él no responden a los procesos que representan nuevos requisitos para el sistema Quarxo, impidiendo que puedan ser utilizadas como referencia para el desarrollo de la solución.

**SAP FOR BANKING.**

Es una aplicación desarrollada por SAP para bancos, utilizada por las entidades financieras para garantizar el control de los procesos críticos de la organización a partir de procesos bancarios específicos. Permite la generación manual y automática de pagos y cobros mediante distintas formas, contiene un asistente que facilita la creación de los mismos por lotes para transferencias bancarias, cheques y letras.[10]

SAP para bancos es un sistema privativo, trayendo como consecuencia el pago de licencias para su uso. No brinda la posibilidad de ver el código fuente para su estudio, impidiendo que se pueda utilizar y adaptar para el desarrollo de las nuevas funcionalidades.

**OPEN ERP.**

Es un sistema de gestión empresarial de código abierto que cubre las necesidades de las áreas de contabilidad, finanzas, ventas, RRHH, compras, proyectos y almacén. Contiene varios módulos oficiales entre los que se encuentra el de Gestión Contable y Financiera, a partir del cual se pueden gestionar las cuentas pendientes de cobros y de pago.[11]

Es un software multiplataforma, funciona sobre Linux y Windows, fue programando en Python y emplea PostgresSQL como sistema manejador de bases de datos.[11] El uso de este gestor de base de datos influye en que no sea tomado como alternativa de estudio para la solución, debido a la utilización de SQL Server para el manejo de la base de datos como requisito fundamental del cliente. Las operaciones sobre los vencimientos que se realizan en el BNC no se corresponden con las realizadas por este sistema.

Una vez realizado el estudio y análisis de los sistemas contables existentes que incluyen de una forma u otra la gestión de los vencimientos entre sus procesos, se arriba a la conclusión de que no existe un sistema altamente configurable que permita el manejo de los cobros y pagos y se adapte a las necesidades propias del BNC.

### 1.3.2 Sistemas nacionales.

**SABIC.**

El SABIC (Sistema automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de Bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado. Permite la contabilización en tiempo real y en varias monedas; y trabaja modularmente.[12] Pero presenta algunas deficiencias como son:

- ✓ Utiliza MS-DOS como sistema de explotación. Este es un sistema operativo antiguo y mono tarea que puede ralentizar y entorpecer el trabajo del operador.
- ✓ No posee integración con los sistemas SLBTR y SISCOM[4]: usados para la comunicación entre los bancos nacionales y con los bancos internacionales, a través de servicios de mensajería.
- ✓ No ofrece el tratamiento requerido a la validación de los datos de entrada impidiendo que sean detectadas de forma temprana ciertas irregularidades.
- ✓ No genera reportes personalizados que puedan ser configurados por el usuario y que muestren información específica sobre la actividad de la entidad.
- ✓ Existe el inconveniente de que los informes deben confeccionarse manualmente por el usuario a través de programas ofimáticos para su posterior impresión, además de que la información almacenada por el sistema no es suficiente para la confección de los mismos.[4]

El SABIC necesita de forma permanente para la explotación del sistema tres ficheros: Maestros Contables, Maestros de Referencia y Maestros Control. Los Maestros Contables están compuestos por los ficheros Histórico, Mayor y Diario, siendo en este último donde se encuentran esperando los vencimientos para ser procesados.

La complejidad y dinamismo de la gestión de los vencimientos en el BNC han provocado que las transacciones del SABIC no respondan eficientemente a sus necesidades operacionales, ralentizando y dificultando la realización de sus operaciones, posibilitando además la alta ocurrencia de errores por parte de los operarios a consecuencia de la carencia de validaciones. Otro factor importante lo constituye el hecho de que no utiliza el mismo sistema de explotación, que las aplicaciones de mensajería

[4]SISCOM: Servicio de transmisión de mensajes financieros

empleadas para la comunicación bancaria tanto nacional como internacional, ni se encuentra integrado a las mismas. Este inconveniente trae como consecuencia que las operaciones sobre los vencimientos que incluyen este servicio, impliquen el constante reinicio de la máquina para realizar la contabilización por un sistema operativo, y la elaboración y envío del mensaje a través de otro, aumentando la posibilidad de deterioro gradual del funcionamiento de las computadoras y disminuyendo el rendimiento de los operadores. La mayoría de las funcionalidades de los vencimientos involucran cálculos complejos que deben realizarse manualmente, además de la entrada de datos de cada asiento para su contabilización, evidenciándose de esta manera la necesidad del desarrollo de un software adaptado a las características del BNC, que garantice la flexibilidad, facilidad de uso y reducción del tiempo y de la probabilidad de ocurrencia de errores en las operaciones. Con este objetivo se crea el sistema Quarxo, sobre el núcleo contable del SABIC.

**QUARXO.**

Quarxo es un sistema informático desarrollado en la UCI por el proyecto SAGEB para garantizar la gestión de los procesos de una forma más sencilla, segura y eficiente, a través de un conjunto de subsistemas. Posee integración con los sistemas de mensajería y ofrece el tratamiento requerido a la validación de los datos de entrada posibilitando la detección de forma temprana de cualquier irregularidad. Posibilita generar reportes por el usuario, que muestran información específica sobre la actividad contable.

Entre los subsistemas que contiene Quarxo se encuentra Vencimiento, que permite a través de un gran número de operaciones, el procesamiento de los compromisos de cobros y pagos que pueden representar negociaciones de cartas de crédito, renegociaciones, préstamos o depósitos, registrados por las áreas correspondientes.

El subsistema tiene como interfaz inicial un buscador genérico que brinda una serie de criterios para el filtrado dinámico de los vencimientos disponibles, que finalmente se muestran en una tabla de resultados. Permite selección múltiple, paginación de resultados, totalización de importes de principal, interés, mora y un global; la entrada de importes parciales, la estimación y cálculo de los importes por concepto de penalización por mora sobre saldos parciales o totales, la impresión de las pre-carteras para los supervisores, e información valiosa tanto para visualización del usuario como para la contabilización de los vencimientos. Muestra un conjunto de acciones que representan las operaciones del proceso, entre las que se encuentran: pago, cobro, renegociación, capitalización y acumulación de intereses. Una vez

desarrollado el subsistema fueron identificadas un conjunto de nuevas funcionalidades dentro del proceso, que no eran soportadas por ninguna de las transacciones existentes en Vencimiento, exigiéndose la utilización de la TGC. Las características de esta transacción pueden complejizar y ralentizar la realización de cada proceso teniendo en cuenta las especificidades de cada uno. Evidenciándose la necesidad de incluir las nuevas funcionalidades en el subsistema Vencimiento, aprovechando de esta manera las ventajas que este brinda.

Para su implementación se hace uso de la misma metodología, arquitectura, herramientas y tecnologías utilizadas en el desarrollo de la primera fase de Quarxo. A continuación se detallan algunas características de cada una de ellas.

## 1.4 Metodología de Desarrollo: RUP.

RUP[5] es una metodología que se utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Está preparada para desarrollar grandes y complejos proyectos como el del BNC, teniendo en cuenta la magnitud, complejidad y dinamismo de sus procesos. Propone una división del trabajo en pequeñas partes que constituyen iteraciones. Como resultado de cada iteración prácticamente se obtiene una versión del producto que debe ser validado con el cliente, permitiendo que sean detectadas y solucionadas de forma temprana sus inconformidades.[13]

Su ciclo de vida se caracteriza por ser:

Dirigido por casos de uso: posibilitando que se refleje lo que los usuarios futuros necesitan y desean, captándose cuando se modela el negocio y se representa a través de los requisitos.[13]

Centrado en la arquitectura: mostrando una visión común del sistema completo, en la que el equipo de proyecto y los usuarios deben estar de acuerdo, describiendo así los elementos del modelo que son más importantes para su construcción. [13]

Iterativo e incremental: permitiendo detectar tempranamente desajustes e inconsistencias entre los requisitos, el diseño e implementación del sistema, de manera que el equipo de desarrollo se mantiene enfocado en producir resultados que satisfagan las necesidades del usuario.[13]

[5] RUP: Proceso Unificado Racional (Rational Unified Process en inglés)

RUP define 6 flujos de trabajo de ingeniería, de ellos se utilizarán en el presente trabajo los Flujos de Trabajo: Requisitos, Análisis y Diseño, Implementación y Pruebas. Los cuales proponen un conjunto de artefactos, que proporcionan los elementos necesarios para la construcción de un software de gran envergadura con la calidad requerida, como es el caso del sistema Quarxo. Por las razones antes mencionadas se decide utilizar esta metodología para el desarrollo de este sistema.

### 1.4.1 Requisitos.

**Técnicas para la captura de requisitos.**

La obtención de los requisitos es una de las etapas fundamentales en el desarrollo de cualquier software. Una adecuada compresión de los requisitos puede contribuir al desarrollo de mejores sistemas que cumplan con las necesidades y expectativas del cliente. Para llevar a cabo este procedimiento se han propuesto varias técnicas que guían al analista en el proceso de establecer una comunicación con el cliente y el equipo de desarrollo.[14]

Las técnicas a utilizar para la obtención de los requisitos funcionales del presente trabajo son:

**Entrevistas.**

Consiste en reuniones analista-interesado en las cuales se suceden preguntas y respuestas para extraer el dominio de la aplicación. En Pressman se presentan conjuntos de preguntas que se pueden utilizar en el desarrollo de esta técnica, que tiene una alta participación del analista.[14]

**Tormenta de ideas (brainstorming).**

Es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de captura, cuando los requisitos son todavía muy difusos.[14]

**Requisitos funcionales y no funcionales.**

Los requisitos para un software son la descripción de los servicios proporcionados por el sistema y sus restricciones. Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares.[15]

**Validación de los requisitos.**

La validación de requisitos tiene como objetivo definir que los requisitos especificados realmente detallan el sistema que el usuario necesita o desea. Verifica que las especificaciones de requisitos no presentan omisiones, conflictos y ambigüedades, además de que sean correctas las interpretaciones por parte del equipo de desarrollo de software.[15] Entre las técnicas utilizadas para validar los requisitos identificados se utilizaron las siguientes.

**Revisiones de requisitos**: Esta técnica se utiliza para corregir cualquier error existente en la documentación o modelado de los requisitos, con el objetivo de encontrar conflictos en el producto, y poder trazar alternativas de solución.[15]

**Construcción de prototipos**: Es utilizado como modelo a escala de la solución final, para brindarle al cliente una visión más clara de cómo quedaría el producto que va a recibir. Mediante los prototipos se puede verificar si las especificaciones han sido construidas de acuerdo a los requisitos del sistema. Teniendo como resultado un modelo para el desarrollo del producto atendiendo a las necesidades específicas del cliente.[15]

### 1.4.2 Análisis y Diseño.

**Modelo de análisis.**

El modelo de análisis describe la estructura de la aplicación que se está modelando, identificando las principales clases y conteniendo un conjunto de realizaciones de casos de uso, que describe cómo el sistema va a ser construido. Tiene como objetivo principal transformar los requisitos en una especificación que describa cómo implementar el sistema, consistiendo fundamentalmente en obtener una visión que permita ver que hace el sistema a desarrollar. Se centra en los requisitos funcionales y se considera una primera aproximación al modelo de diseño.[13]

**Diagrama de Clases del Análisis.**

El diagrama de clases del análisis es un diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Estos diagramas de clases son utilizados durante el proceso de análisis y el diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargan del funcionamiento y la relación entre uno y otro centrándose en el tratamiento de los requisitos funcionales.[13]

**Modelo de diseño.**

El modelo de diseño constituye un modelo de objetos que describe la relación física existente entre los CU. A partir de los requisitos se definen los casos de uso y se genera el modelo de diseño que incluye los artefactos: diagrama de clases del diseño, diagrama de paquetes, y diagramas de interacción. El modelo de diseño es utilizado como una abstracción de la implementación del sistema y como una entrada fundamental de la actividad de implementación.[13]

**Diagrama de Clases del Diseño.**

Un diagrama de clases del diseño describe la estructura de un sistema mostrando sus clases, atributos y sus relaciones entre sí. Estos diagramas se utilizan durante el proceso de análisis y diseño de los sistemas, para crear el diseño conceptual de la información que se manejará, los componentes que se encargarán del funcionamiento y sus relaciones.[13]

**Validación del diseño.**

El diseño constituye el punto de partida para el desarrollo del sistema una vez especificados los requisitos, y es el encargado de convertirlos en un modelo que al ser implementado, se obtenga el producto deseado por el cliente. Por ello realizar una validación del mismo es importante para garantizar su calidad, para la implementación. Para la validación del diseño se utilizan las métricas: Tamaño Operacional de Clases y Relaciones entre Clases.[16]

**Tamaño Operacional de Clases (TOC).**

Esta métrica es utilizada para evaluar los atributos de calidad: Responsabilidad, Complejidad de implementación y Reutilización, determinada por el número de métodos asignados a una clase. De esta forma mientras menos sea el número medido, menor será la responsabilidad y complejidad que posea la clase y mayor su nivel de reutilización.[16]

**Relaciones entre Clases (RC).**

Esta métrica se utiliza con el fin de evaluar los atributos de calidad: Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de pruebas, siendo determinada por el número de relaciones entre clases. Un aumento de las RC implica un aumento directo en el acoplamiento, la complejidad de

mantenimiento y la cantidad de pruebas en las clases y una disminución en la reutilización de las mismas.[16]

### 1.4.3 Implementación.

**Modelo de Implementación.**

En el flujo de trabajo de implementación es donde se realiza la solución a partir del diseño definido, en él se describe cómo los elementos del Modelo de Diseño se implementan en términos de componentes y cómo estos se organizan en nodos específicos en el Modelo de Despliegue.[16]

**Diagrama de Componentes.**

Un diagrama de componentes es una ilustración de los elementos genéricos e independientes que conforman un software. Poseen un nivel de abstracción más elevado que los diagramas de clases, debido a que un componente usualmente está compuesto por más de una clase.[16]

### 1.4.4 Pruebas.

Las pruebas son un conjunto de actividades que permiten verificar y validar la calidad de un software. Estas se le realizan a un sistema bajo condiciones y requisitos específicos, donde los resultados son observados y documentados para dar una evaluación del producto y determinar la calidad del mismo.[16] Debido al crecimiento en complejidad y tamaño de las aplicaciones de software, es necesario que el proceso de pruebas se realice con la calidad y eficiencia requerida, logrando así la aceptación final por parte del usuario. Los métodos a utilizar para las pruebas son: Caja Blanca y Caja Negra los cuales se adecuan al nivel de prueba Unidad. Este nivel se enfoca en el código fuente de los componentes verificando de forma individual las partes del sistema que han sido desarrolladas.

**Caja Blanca:**

Las pruebas de caja blanca son aquellas que se realizan sobre las funciones internas de un software o una porción del mismo. Se basa en el diseño de Casos de Prueba atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento. Además de realizar un seguimiento del código fuente según se van ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones en las que existen errores.[16]

**Caja Negra.**

Estas pruebas se centran principalmente en los requisitos funcionales del software, es decir, mediante los casos de prueba se pretende demostrar que las funciones del sistema son operativas. Las pruebas de caja negra se realizan mediante la introducción de valores de entrada, los cuales deben ser aceptados adecuadamente y producir una salida correcta. Estos se realizan sin preocuparse de lo que pueda estar haciendo el software internamente, debido a que todas las pruebas se realizan sobre la interfaz de usuario.[16]

La prueba de Caja Negra intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca, siendo un enfoque complementario y no una alternativa a las técnicas de prueba de la Caja Blanca [16]. Mediante ella se intentan encontrar:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.[16]

## 1.5 Patrones arquitectónicos.

Los patrones arquitectónicos son patrones de alto nivel encargados de especificar un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes, determinando la arquitectura global de la aplicación. Facilitan la construcción de un software mediante un vocabulario común, y ayudan a construir arquitecturas heterogéneas y complejas.[17] La arquitectura de Quarxo aplica el patrón MVC.

**MVC (Modelo Vista Controlador):** El patrón de arquitectura Modelo-Vista-Controlador (MVC) divide una aplicación interactiva en tres partes. El modelo contiene los datos y la funcionalidad esencial. Las vistas despliegan la información al usuario. Los controladores manejan las entradas. Las vistas y los controladores juntos componen la interfaz con el usuario. El mecanismo de cambio-propagación asegura la consistencia de la interfaz con el modelo.[18]

## 1.6 Patrones de diseño.

Un patrón de diseño define un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos. Este describe una estructura común y recurrente de componentes interrelacionados, que resuelve un problema general de diseño dentro de un contexto particular.[19]

A continuación se describen un conjunto de patrones que contribuyen al buen diseño del subsistema, tales como:

**DAO (Patrón de acceso a datos).**

Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos. Engloba todo el acceso a datos en una capa independiente, desacoplando la lógica de negocio de la lógica de acceso a datos de manera que se pueda cambiar la fuente de datos fácilmente.[20]

**Patrones de asignación de responsabilidades (GRASP).**

**Patrón experto.**

Consiste en asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.[19]

**Patrón creador.**

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.[19]

**Patrón controller.**

Asignar la responsabilidad del manejo de los eventos de un sistema a clases específicas que cumplan la función de intermediarias entre la interfaz de usuario y las clases donde reside la lógica de la aplicación.[19]

**Patrón bajo acoplamiento.**

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Proponiendo la baja dependencia entre clases, para que el código sea de fácil entendimiento, mantenimiento y posibilitando la fácil reutilización del mismo.[19]

**Patrón alta cohesión.**

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, logrando un alto grado de funcionalidad combinada con una reducida cantidad de operaciones.[19]

**Patrones estructurales GOF.**

- ✓ **Facade (Fachada):** El patrón Fachada proporciona una interfaz unificada de alto nivel para un subsistema, que oculta las interfaces de bajo nivel de las clases que lo implementan. Con esto se consiguen dos objetivos fundamentales: hacer el subsistema más fácil de usar y desacoplar a los clientes de las clases del subsistema.[19]
- ✓ **Composite View (Vista compuesta):** Crear Vistas Compuestas de varias sub-vistas de forma modular, flexible y extensible para construir vistas de páginas JSP y HTML usando la directiva include.[21]

## 1.7 Ambiente de desarrollo.

Para el desarrollo de la segunda fase del subsistema Vencimiento del sistema Quarxo se utilizan un conjunto de herramientas, tecnologías y lenguajes, que fueron seleccionadas con anterioridad en el desarrollo de la primera fase, para lograr la compatibilidad y uniformidad en la aplicación. A continuación se realiza una breve descripción de cada una de ellas:

### 1.7.1 Lenguajes y herramientas.

**UML.**

Unified Modeling Language (UML) es ante todo un lenguaje, esencial en la construcción de software para comunicar la estructura de un sistema complejo, especificar el comportamiento deseado del sistema, comprender mejor lo que se está construyendo y descubrir oportunidades de simplificación y reutilización.[19]

**Java.**

Java es un lenguaje de programación orientado a objeto, multiplataforma y de código abierto. Se caracteriza por ser un lenguaje simple, robusto y poderoso, contando con un gran número de librerías que facilitan el trabajo en el tratamiento de excepciones, procesamientos concurrentes y trabajos con la red. (24). Este lenguaje es el que se utiliza para el desarrollo del sistema Quarxo.[22]

**Javascript.**

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. JavaScript es un lenguaje de programación interpretado, no siendo necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. Lo soportan la mayoría de los navegadores como son: Netscape, Firefox e Internet Explorer.[23]

**SQL (Standard Query Language).**

Es un lenguaje declarativo de acceso a bases de datos relacionales y de alto nivel que permite especificar diversos tipos de operaciones sobre las mismas. Además realiza funciones de definición, control y gestión de la base de datos.[24]

**HQL (Hibernate Query Language).**

Es un lenguaje que posee una extensa documentación, ofrece una gran potencia para la obtención de los datos a partir de una base de datos, a través de las consultas que se realizan sobre los objetos y sus propiedades, y no sobre las tablas directamente, posibilitando la simplificación del código y desacoplando el uso de una base de datos en particular. Funciona como intermediario, que a partir de la base de datos y el dialecto especificado permite que las consultas en HQL del mundo objetual sean traducidas al relacional SQL de una forma transparente y automática.[25]

**Plataforma JEE.**

Como plataforma de desarrollo se decide utilizar Java Enterprise Edition (JEE), la cual es la edición empresarial de la plataforma Java. Aprovecha las fortalezas de la edición estándar de Java (J2SE), complementándolas con especificaciones, funcionalidades y lineamientos orientados al desarrollo de aplicaciones empresariales.[26]

**Contenedor Web (Tomcat 6.0).**

Apache Tomcat funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets 2.5 y de JavaServerPages (JSP) 2.1. Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.[27]

**IDE de desarrollo (Eclipse 3.3).**

Se seleccionó como entorno de desarrollo el eclipse en su versión 3.3 por ser una plataforma libre, de código abierto y presentar una arquitectura de plugins que lo hace flexible y configurable, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.[28]

**Visual Paradigm 3.4.**

Esta herramienta permite el modelado del ciclo completo de RUP, así como generar cualquier tipo de diagrama[29]. La UCI cuenta con la licencia de esta herramienta lo cual hace aún más acertada su elección.

**Erwin.**

Esta herramienta se utiliza para el modelado de la base de datos en su versión 7.5, permitiendo diseñar, mantener y generar bases de datos desde un modelo lógico hasta un modelo físico.[30]

**Control de versiones (SubVersion).**

Está desarrollado sobre tecnología Open Source y se integra con Eclipse mediante el plugin SubEclipse. Permite recuperar versiones antiguas de los datos registrados y examinar el historial de los mismos, manejando la información de ficheros y directorios a través del tiempo, la cual radica en un árbol de ficheros en un repositorio central.[31]

**Base de Datos (SQL).**

Se utiliza el SQLServer 2005 para la gestión de la base de datos por petición del BNC y el BCC, con el objetivo de ganar en tiempo en el desarrollo del sistema con la reutilización del núcleo del SABIC. Esta herramienta necesita para su funcionamiento del sistema operativo Windows, atentando de esa manera contra la premisa del desarrollo de los sistemas para nuestra sociedad sobre plataforma de software libre.

Se caracteriza por brindar soporte para transacciones y procedimientos almacenados y posee como lenguajes de consulta al SQL y al T-SQL (en inglés: Transact-SQL).[32]

### 1.7.2 Frameworks.

**Spring.**

Es un framework de código abierto orientado al desarrollo de aplicaciones para la plataforma Java. Fue creado por Rod Johnson, quien lo describió por primera vez en su libro "Expert One-on-One Java EE Design and Development". Es el más popular y el más ambicioso de todos los framework de peso ligero. Es el único framework que interviene en todas las capas arquitectónicas de una aplicación JEE: acceso a datos, negocio y presentación. Además está diseñado para facilitar una flexibilidad arquitectónica.[33]

- ✓ **Spring Web.**

Es el encargado de crear el contexto para aplicaciones web, incluye soporte para una variedad de tareas como: la carga de archivos y la vinculación de parámetros de las peticiones a objetos del negocio.[34]

- ✓ **Spring *WebFlow.***

Flujo se puede definir como una secuencia de pasos o actividades que se realizan para llevar a cabo una determinada acción. Spring WebFlow es el módulo de Spring para implementar cualquier tipo de flujo de una manera clara. Se acopla con la plataforma de Spring Web MVC y provee una definición de un lenguaje declarativo de flujos. Se utiliza cuando el caso de uso (CU) que se desarrolla presenta un flujo complejo y/o no lineal. La configuración del desarrollo del flujo es especificada a través de un archivo de configuración XML, permitiendo establecer reglas de navegación complejas de una manera sencilla. Un flujo en SpringWebFlow maneja la conversación (vacío entre Sesión y Petición) completa, de inicio a fin. Se hace uso de Spring WebFlow 2.0.8.[34]

- ✓ **Spring MVC.**

Se utiliza para atender las peticiones web simples con navegación lineal a través de clases **Controller.** Entre las que se encuentran las clases SimpleFormControler y MultiActionControler. Las cuales son usadas para trabajar con un conjunto de parámetros recibidos desde una petición, permitiendo que el formulario que contiene dichos parámetros se cargue inicialmente generando la página donde serán

procesados los datos y la atención a múltiples peticiones, permitiendo el procesamiento de las mismas, estas generalmente son de un negocio determinado.[34]

✓ **Spring DAO.**

Posee algunas librerías de clases para trabajar con base de datos a través del API de Java JDBC y brinda soporte para invocar procedimientos y funciones almacenadas. Ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos. Ayuda a mantener el código simple y evita que sea muy repetitivo, además minimiza los errores al intentar cerrar la conexión con algunas bases de datos.[34]

✓ **Spring ORM.**

Spring ORM (Object Relation Mapping) es un módulo de Spring que permite su integración con los frameworks ORM más populares en el mercado. Dicha integración proporciona mayor seguridad, facilidad y eficiencia en el manejo de sesiones, recursos, transacciones ORM, excepciones y pruebas. Es utilizada en el proyecto para lograr la integración de Spring con el framework ORM Hibernate.[34]

✓ **Spring Transaction.**

Para garantizar la ejecución de forma transaccional de las funcionalidades de la aplicación se decide utilizar Spring Transaction. Soporta el manejo de las transacciones para varios gestores de base de datos y recursos compartidos a través del API JTA[6], Spring Transaction permite definirlas en forma de anotaciones o basándose en declaraciones AOP[7] en el código de la aplicación, y exponerlas en los ficheros declarativos. Esta última forma es la más recomendaba en la mayoría de los casos, debido a que la administración de las transacciones requiere menos código y por tanto no depende de ninguna API.[34]

✓ **Spring Security.**

Permite gestionar completamente la seguridad de aplicaciones java, ofreciendo servicios de identificación de usuarios y acceso a los recursos sin añadir código; proporciona asignación de permisos a usuarios o grupos de usuarios. Separa claramente la seguridad de un sistema de su lógica de negocio, haciendo más fácil y eficiente su administración.[34]

[6] Java Transaction API.

[7] Aspect Oriented Programming.

**Hibernate.**

Hibernate es un framework objeto/relacional y un generador de sentencias SQL. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. Hibernate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos. Permite generar las sentencias y ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución, ofrece también un lenguaje de consulta de datos llamado HQL , al mismo tiempo que una API para construir las consultas programáticamente, conocida como "Criteria".[25]

**Dojo 1.3.**

Es una biblioteca JavaScript de código abierto, la cual brinda una variedad de clases y widgets para facilitar el desarrollo de aplicaciones web. Dojo puede ser interpretado por diferentes navegadores web y posee un sistema de empaquetado muy parecido al del JDK de java. La gran variedad de clases componentes y widgets, es un punto muy importante en la elaboración en la capa presentación de la aplicación, posibilita a los programadores de interfaz una serie de funcionalidades y elementos dinámicos que facilita la programación en el cliente. Incluye varios API, efectos visuales, un gran cúmulo de componentes visuales basados en XHTML, CSS y JavaScript que permiten el desarrollo de aplicaciones web enriquecidas.[35]

**JUnit.**

Su objetivo principal es evaluar el funcionamiento de cada uno de los métodos dentro de las clases que conforman el software. Este framework permite la automatización de las pruebas de aplicaciones en Java, consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba y ejecutarlos automáticamente. Además mejora el diseño de implementación, haciéndolo flexible y testeable.[36]

## 1.8 Conclusiones Parciales.

A partir de un estudio realizado de los sistemas informáticos bancarios nacionales e internacionales que incluyen dentro de sus funcionalidades el manejo de los vencimientos, se concluye que los mismos no representan soluciones al problema a resolver y se evidencia la importancia del desarrollo de software bancario cubano. Quarxo, ejemplo de solución interna a nuestras necesidades, incluye un subsistema

(Vencimiento) que garantiza en parte el trabajo con los vencimientos en el BNC, pero algunas de las operaciones de este proceso no están incluidas dentro de sus funcionalidades, ralentizando y complejizando la ejecución de las mismas. Una segunda fase de Vencimiento que incluya esas operaciones es la solución factible al problema existente. Teniendo en cuenta las características del entorno de desarrollo y el de despliegue y la necesidad de reducción del tiempo de desarrollo y esfuerzo de trabajo, en la creación de una segunda fase del subsistema se hace uso de las mismas herramientas y tecnologías seleccionadas para la obtención de la primera.

# CAPÍTULO 2: REQUISITOS Y ANÁLISIS DE LA SOLUCIÓN.

## 2.1 Introducción.

Como punto de partida en el proceso de desarrollo de software siguiendo la metodología RUP, se lleva a cabo un modelado del negocio donde se describen los procesos existentes con el objetivo de comprenderlos. Para el desarrollo de la solución no es necesario realizar un modelado de negocio, debido a que se tiene un conocimiento previo de la estructura de la organización, de los clientes, la visión y de las metas, además de que los flujos de trabajo de la entidad están bien documentados. Por esta razón se comienza la investigación a partir del análisis de los requisitos, que incluye una representación detallada de las funcionalidades, que fueron identificadas dentro del proceso de gestión de los vencimientos una vez desarrollada la primera fase. El contenido de la sección incluye los requisitos funcionales y una descripción de los casos de uso que responden a los mismos, así como especificidades de los requisitos no funcionales señalados.

## 2.2 Técnicas para la captura de los requisitos funcionales.

Para la captura de los requisitos funcionales del presente trabajo se utilizaron dos técnicas, tales como:

- ✓ **Entrevistas:** se realiza a través de reuniones efectuadas con los clientes para obtener la información necesaria de los procesos de gestión de los vencimientos en el BNC, que no están incorporados en el subsistema Vencimiento. A partir de las cuales se obtuvo un resultado positivo permitiendo de esta manera identificar todas las necesidades que poseía el cliente
- ✓ **Tormenta de ideas:** para su puesta en práctica fue necesario realizar una reunión en la cual los clientes y el equipo de desarrollo brindaban sus ideas en cuanto a la solución propuesta.

## 2.3 Descripción de los requisitos funcionales y no funcionales.

En la tabla 1 se muestran los nuevos requisitos funcionales identificados dentro del proceso de gestión de los vencimientos.

| **Requisitos funcionales** | |
|---|---|
| **RF1**-Registrar vencimientos | **RF7**-Cobrar comisiones a cuenta única |

| | |
|---|---|
| **RF2**-Actualizar vencimientos | **RF8**-Registrar importación. |
| **RF3**-Actualizar renegociación | **RF9**-Registrar exportación. |
| **RF4**-Migrar calendario | **RF10** Rebaja de importación/exportación. |
| **RF5**-Pago agrupado de vencimientos | **RF11** Pagar comisión. |
| **RF6**-Condonar vencimientos | |

Tabla 1: Requisitos funcionales

**Registrar vencimientos:**

Consiste en registrar en el registro M_DIARIO vencimientos de intereses moratorios, de principal y de intereses ordinarios, con sus respectivas contrapartidas almacenadas en la configuración. En caso deseado los vencimientos de principal y de intereses ordinarios pueden asociarse a un calendario existente, donde serán actualizados los importes correspondientes, en caso de que la fecha del vencimiento esté fuera de los límites del calendario se redefinen sus fechas de inicio o fin. Además debe admitir una totalización de cuentas tanto carteras como no carteras.

**Actualizar vencimientos:**

Consiste en la redefinición de un conjunto de datos de vencimientos contabilizados en el sistema que no formen parte de ningún calendario.

**Actualizar renegociación:**

Consiste en la redefinición de determinados datos de una renegociación del operativo, que pueden no representar una información contable como: el nombre de la renegociación, o que si la requieren, ya sean datos del operativo de deuda o propios del calendario.

**Cobrar comisiones a cuenta única:**

Consiste en la selección y extracción de todos los vencimientos por cobrar hasta una fecha determinada que representan comisiones y que son obligaciones de cuenta única; y la creación de un nuevo

vencimiento por el total, además se envía un mensaje SLBTR a cuenta única solicitando los fondos que debe pagar por conceptos de estas comisiones.

**Condonar vencimientos:**

Consiste en rebajar un por ciento determinado al importe de vencimientos contabilizados en el sistema.

**Migrar calendario:**

Esta funcionalidad surge por la necesidad de actualizar el operativo de calendario con los vencimientos correspondientes del registro M_DIARIO, necesaria para la ejecución de la mayoría de las funcionalidades del subsistema una vez efectuada una carga inicial de la base de datos. De manera opcional debe permitir la actualización de otros operativos como el de negociación y el de renegociación, importante para el logro de la integridad y consistencia de datos en los reportes correspondientes que se generan automáticamente.

**Pago agrupado de vencimientos:**

Consiste en la realización de múltiples pagos de vencimientos contabilizados en el sistema de forma tal que en un único pago se puedan tratar diferentes orígenes de fondos, ejemplo: fondos propios, cuenta única y cuentas por cobrar, agrupados por diferentes criterios como: referencia original y fecha de vencimiento, contra uno o varios créditos a una cuenta banco. Se integra a la funcionalidad pago de vencimiento por las facilidades que brinda.

**Registrar importación.**

Consiste en el registro de los intercambios comerciales y de pago entre la República Popular de Corea y Cuba para la compensación de productos industriales. Donde la importación de productos industriales por parte de Cuba es compensada con la exportación de azúcar.

**Registrar exportación.**

Consiste en el registro de los procesos de intercambio comercial y de pago entre la República Popular de Corea y Cuba para la compensación de productos industriales. Donde a partir de la exportación de azúcar por parte de Cuba, se recibe a cambio otros tipos de productos.

**Rebaja de importación/exportación.**

Sería el contravalor[8] que se le paga a la empresa exportadora por la mercancía importada por la otra empresa (importadora) bajo una moneda de convenio que no es real (USS).

**Pagar comisión.**

Consiste en el cobro de una comisión fija por parte del BNC de 150 CUC a las empresas importadoras y exportadoras por cuestiones de trámites.

## 2.4 Diagrama de casos de uso del sistema.

Uno de los principales artefactos del flujo de trabajo de Requisitos es el diagrama de casos de uso del sistema, que representa gráficamente a los procesos y su interacción con los actores del sistema. Este diagrama proporciona una visión global de los casos de uso de sistema identificados.

### 2.4.1 Descripción de los actores del sistema.

Los actores del sistema son los que interactúan directamente con el mismo. Teniendo en cuenta que Vencimiento se encuentra integrado con un subsistema de Seguridad, en el diagrama que muestra la Figura 1 los actores del sistema responden a roles ya definidos, que contienen un conjunto de permisos y restricciones de acceso que garantizan la seguridad del subsistema. Los nombres de estos roles sugieren el área operaria que realiza el proceso correspondiente. Los procesos “Registrar vencimientos”, “Actualizar vencimientos”, “Cobrar comisiones a cuenta única” y “Migrar Calendario” pueden ser realizados por usuarios de cualquier área operativa. Por esta razón en el diagrama se representa una generalización y especialización de actores a partir del actor principal que interactúa con el sistema: el especialista. Este usuario representa cualquier operario del banco, debido a que el subsistema Vencimiento es usado por todas las áreas de la entidad. Este actor interactúa con los casos de usos que pueden ser realizados por cualquier operario independientemente del departamento al que pertenezca (registrar y actualizar vencimientos, cobrar comisiones a cuenta única y migrar calendario). Los demás actores que intervienen en el sistema heredan las relaciones del actor principal con los casos de uso mencionados, además interactúan con otros casos de uso según el área a la que pertenezcan, estos son: Esp_Negociación que efectúa “Pago agrupado de vencimiento”, “Registrar importación”, “Registrar exportación”, “Rebaja de importación/exportación” y “Pago de comisión”; Esp_Renegociación: “actualizar y condonar vencimiento”;

8 Precio o valor que se da a cambio de lo que se recibe.

además de los casos de uso generales. A continuación se muestra el diagrama de casos de uso del sistema de la segunda fase de Vencimiento.

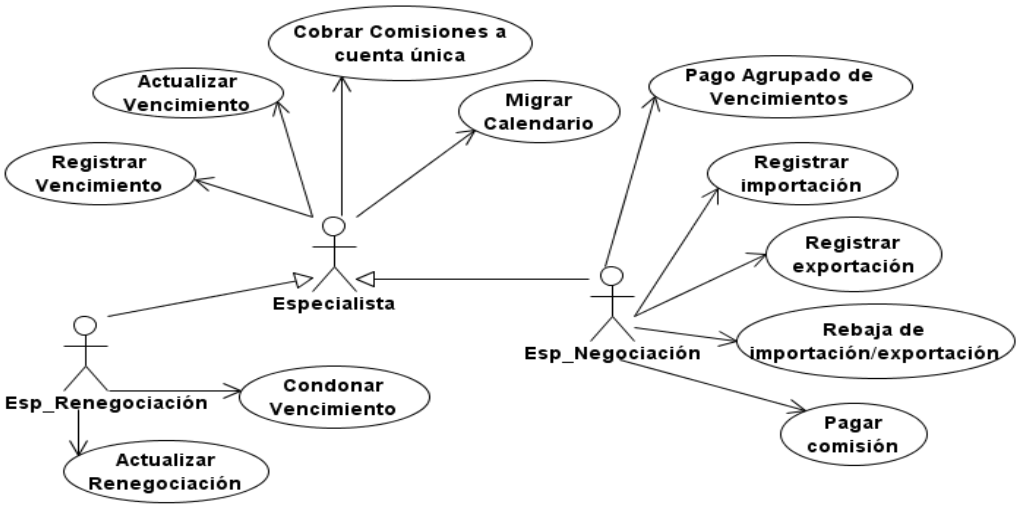


Figura 1: Diagrama de caso de uso del sistema.

## 2.5 Descripción de los casos de uso del sistema.

Teniendo en cuenta que cada uno de los requisitos funcionales mencionados constituye una operación independiente y con una lógica de negocio en particular, se identifica un caso de uso del sistema por cada requisito. Los casos de uso del sistema permiten capturar el comportamiento deseado del sistema sin tener que especificar cómo se implementa ese comportamiento. Representan un medio de comprensión del sistema para desarrolladores, usuarios finales y expertos del dominio.

A continuación se muestra una descripción del caso de uso Registrar vencimiento, tomado como caso de estudio para la visualización en el documento de los principales artefactos de cada flujo de trabajo.

**CU: Registrar Vencimientos.**

| | |
|---|---|
| **Caso de Uso:** | Registrar Vencimientos. |
| **Actores:** | Especialista. |
| **Resumen:** | Permite registrar vencimientos de principal, y de intereses tanto ordinarios como moratorios en el sistema. |

| | |
|---|---|
| **Precondiciones:** | El usuario debe estar autenticado previamente. |
| **Referencias:** | RF1 |
| **Flujo normal de eventos** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| 1- Seleccionar la opción registrar vencimiento del menú principal del subsistema. | 2- El sistema muestra el formulario registrar vencimiento, que incluye la referencias de origen, la externa, fecha contable y fecha valor, así como una tabla donde serán mostrados los vencimientos registrados. Existen las opciones de registrar vencimientos (principal, moratorio u ordinario) o eliminarlos.<br>Además de la opción visualizar asientos contables. **Flujo Alterno 1**- Visualizar asientos |
| 3– El usuario llena los campos vacíos. | 4- El sistema verifica que los datos estén correctos. Si hay datos incorrectos, **Flujo Alterno 2** .Datos incorrectos. |
| 5- El usuario selecciona qué tipo de vencimiento quiere registrar. | 5.1- Si selecciona principal ir a **Sección** Registrar Vencimiento Principal.<br>5.2- Si selecciona interés ordinario ir a **Sección** Registrar Vencimiento Interés Ordinario.<br>5.3- Si selecciona moratorio ir a **Sección** Registrar Vencimiento Moratorio.<br>5.4- Si selecciona eliminar ir a **Sección Eliminar Vencimiento.** |
| 6- Se selecciona aceptar | 7-Se valida que no existan campos vacíos y que no haya errores de negocio. Si existen **Flujo Alterno 3** Campos vacíos.<br>Se contabilizan los asientos registrados. |
| | 8- El caso de uso se termina. |

| **Sección:** Registrar Vencimiento Principal. | |
|---|---|
| **Flujo normal de eventos** | |
| **Acción de Actor** | **Respuesta del sistema** |
| 1- Selecciona registrar vencimiento principal. | 2- El sistema muestra el formulario para introducir los datos del vencimiento a registrar. |
| 3- El especialista introduce los datos. | 4- Se validan los datos. Si son incorrectos. **Flujo Alterno 2,** Datos incorrectos. Y si hay campos vacíos **Flujo Alterno 3** Campos vacíos. |
| 5- El usuario selecciona la opción aceptar | 6- Se registra el vencimiento y se visualiza en la tabla del formulario principal. |
| | 7- Finaliza la acción. |
| **Sección:** Registrar Vencimiento Interés Ordinario | |
| **Flujo normal de eventos** | |
| **Acción de Actor** | **Respuesta del sistema** |
| 1- Selecciona registrar vencimiento interés ordinario. | 2- El sistema muestra el formulario para introducir los datos del vencimiento a registrar. |
| 3- El usuario introduce los datos. | 4- Se validan los datos. Si son incorrectos. **Flujo Alterno 2,** Datos incorrectos. Y si hay campos vacíos **Flujo Alterno 3** Campos vacíos. |
| 5- El usuario selecciona la opción aceptar | 6- Se registra el vencimiento y se visualiza en la tabla del formulario principal. |
| | 7- Finaliza la acción. |
| **Sección:** Registrar Vencimiento Moratorio | |
| **Flujo normal de eventos** | |
| **Acción de Actor** | **Respuesta del sistema** |
| 1- Selecciona registrar vencimiento moratorio. | 2- El sistema muestra el formulario para introducir los |

| | |
|---|---|
| | datos del vencimiento a registrar. |
| 3- El usuario introduce los datos. | 4- Se validan los datos. Si son incorrectos. **Flujo Alterno 2,** Datos incorrectos. Y si hay campos vacíos **Flujo Alterno 3** Campos vacíos. |
| 5- El usuario selecciona la opción aceptar | 6- Se registra el vencimiento y se visualiza en la tabla del formulario principal. |
| | 7- Finaliza la acción. |
| **Sección:** Eliminar Vencimiento | |
| **Flujo normal de eventos** | |
| **Acción de Actor** | **Respuesta del sistema** |
| 1- El usuario selecciona el vencimiento que desea eliminar. | 2- Se elimina el vencimiento seleccionado. |
| | 3- Finaliza la acción. |
| **Flujo Alterno 1:** Visualizar asientos | |
| **Acción del actor** | **Respuesta del sistema** |
| 1- Selecciona la opción mostrar asientos. | 2- Si existen vencimientos registrados en la tabla del formulario principal, se mostrarán en una ventana, los asientos que serán contabilizados, si no hay **Flujo Alterno 4**, No hay asientos para contabilizar. |
| **Flujo Alterno 2:** Datos incorrectos | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1- El sistema muestra un mensaje referente al error que se está cometiendo. |
| **Flujo Alterno 3:** Campos vacíos. | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1- Se señalan los campos vacíos. |

| **Flujo Alterno 4:** No hay asientos para contabilizar. | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| | 1- Se muestra un mensaje: "No hay asientos para contabilizar" y no se muestra la ventana con los asientos. |

Tabla 2: Descripción del caso de uso Registrar Vencimiento.

A continuación se muestran las descripciones de los requisitos no funcionales identificados en la primera fase a través de entrevistas con el cliente, teniendo en cuenta las características del entorno de desarrollo y el de despliegue. La segunda fase del subsistema debe cumplir con estos requisitos.

**Requisitos no funcionales.**

| **Tipo de requisito** | **PCs clientes** | **PCs servidores** | |
|---|---|---|---|
| | | **Servidor 1** | **Servidor 2** |
| **Software** | Máquina virtual de Java 6.0u20 o superior.<br>Morzilla Firefox 3.6.+ | Windows Server 2003.<br>Apache Tomcat 6.0 o superior.<br>Máquina virtual de Java 6.0u20 o superior. | Windows Server 2003.<br>Microsoft SQL Server 2005 o superior. |
| **Hardware** | Procesador Pentium IV o superior 2.0 GHZ o superior.<br>RAM: 256 MB(recomendado 512)<br>Una tarjeta de red. | Procesador: Core 2 Duo 2.0 GHZ o superior<br>RAM: 4 GB<br>Disco duro: 160 GB<br>UPS: 1<br>Lector de CD: 1 | |

Tabla 3: Requisitos no funcionales de software y hardware.

**Funcionalidad.**

El sistema debe mostrar los errores en forma de mensaje, incluyendo una descripción detallada del error.

**Usabilidad.**

Bajo la premisa de que los formularios deben estandarizarse, los campos de texto deben tener un tamaño adecuado con respecto a las dimensiones que se tengan en la página.

Los resultados de las consultas que posean más de las coincidencias permitidas por la tabla en las que se mostrarán, deben ser paginados. Se debe mostrar en la parte inferior de la tabla el total de elementos de la búsqueda encontrados. Es necesario mostrarse en la parte inferior de la tabla opciones de navegación: ir a la primera página, ir hacia atrás, ir hacia la siguiente e ir hacia la última página.

**Disponibilidad.**

El sistema debe estar en uso durante toda la jornada laboral.

**Seguridad.**

Es preciso que el sistema conceda el acceso a partir de un usuario y una contraseña, y que solo permita el acceso de cada usuario a las funcionalidades que se les definieron a partir de su área de trabajo. Se solicita el acceso nuevamente al usuario si se encuentra inactivo durante un tiempo determinado.

**Rendimiento.**

Las operaciones que impliquen un elevado nivel de procesamiento en la base de datos deben usar procedimientos almacenados.

**Restricciones de diseño.**

El estilo arquitectónico del sistema debe estar basado en capas.

**Interfaz de usuario.**

Es necesario que todos los textos y mensajes en pantalla se muestren en idioma español. Los mensajes de error referentes a la inserción errónea de algún dato, deben ser visibles para los usuarios.

## 2.6 Validación de Requisitos.

A partir de una definición previa de los requisitos funcionales se realiza la validación de los mismos, asegurando de esta manera que los resultados obtenidos y el análisis realizado en la etapa de captura son correctos. La técnica utilizada para la validación de los requisitos identificados es la siguiente:

**Prototipos:** Se elaboran prototipos de interfaz de usuario por cada uno de los requisitos identificados. A partir de ellos el cliente tiene una vista previa de cada transacción, que incluye toda la información necesaria tanto de entrada como de salida. Permiten verificar si se satisfacen completamente las necesidades del cliente.

La aceptación por parte del cliente de los prototipos realizados, demostró que la información transmitida a los analistas había sido interpretada correctamente. A continuación se muestra el prototipo de interfaz de usuario del caso de uso Registrar Vencimientos como ejemplo de esta técnica de validación de requisitos.

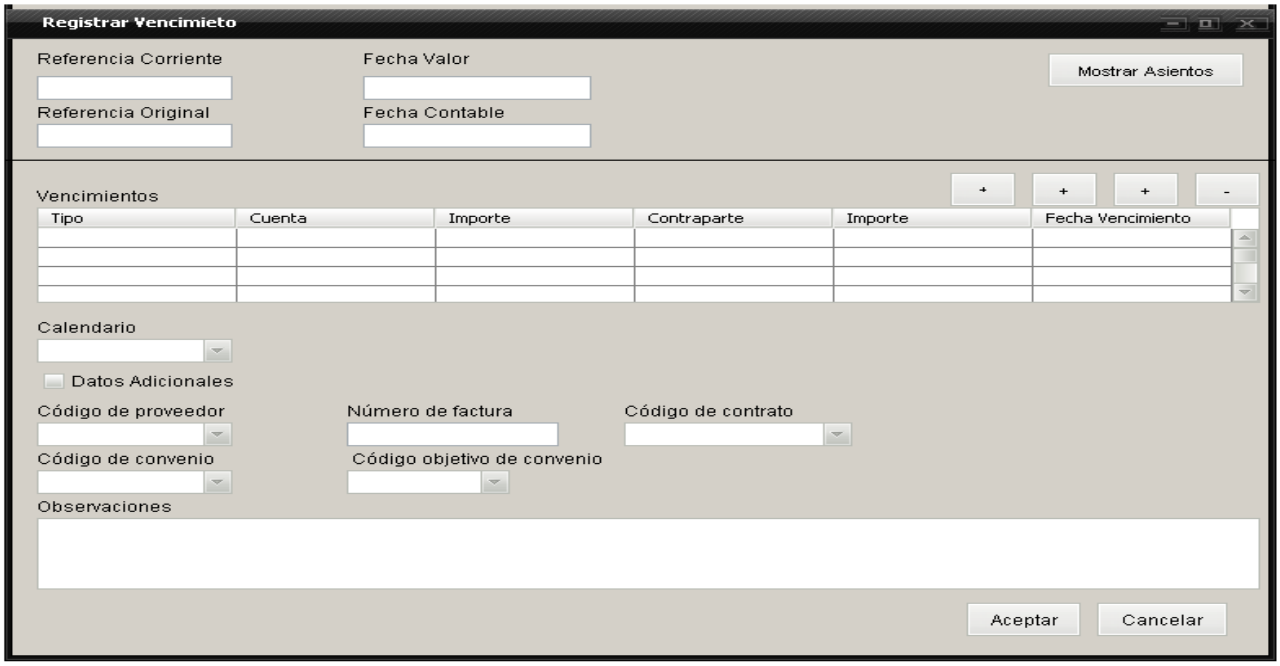


Figura 2: Prototipo de interfaz de usuario Registrar Vencimientos.

Como otro flujo de trabajo de RUP y posterior al de Requisitos se encuentra el de Análisis y Diseño. Específicamente en el análisis se generan un conjunto de artefactos necesarios e imprescindibles para comprender con más claridad la aplicación a desarrollar, como es el caso de los diagramas de clases del análisis que se describirán a continuación.

## 2.7 Modelo de análisis.

Con el desarrollo del modelo de análisis se obtuvo una descripción de la estructura de la aplicación que se está modelando, obteniéndose las principales clases y las realizaciones entre casos de uso, que describen cómo el sistema va a ser construido. Para la descripción de la estructura del sistema se realizan los diagramas de clases del análisis, tomándose como caso de estudio el caso de uso registrar vencimientos.

En la metodología RUP se establecen una serie de estereotipos para las clases del análisis, los cuales se muestran a continuación:

La clase interfaz es la encargada de la interacción entre los actores y el sistema:

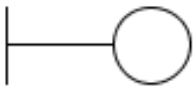

Figura 3 Clase interfaz.

La clase controladora que son las encargadas de manejar y almacenar la lógica de la aplicación:

Figura 4 Clase Controladora.

La clase entidad se encarga de contener la información que será persistida durante un largo tiempo en la base de datos, modelados en el mundo objetual como clases entidades:

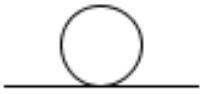

Figura 5 Clase Entidad.

### 2.7.1 Diagrama de Clases del Análisis.

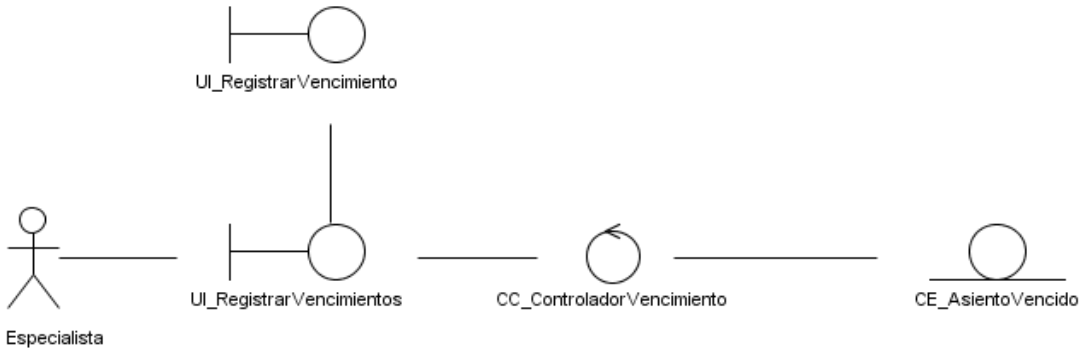


Figura 6 Diagrama de Clase del análisis Registrar Vencimiento.

En este diagrama se muestra la interacción del Usuario con la interfaz principal UI_RegistrarVencimientos, que contiene varios datos generales de la transacción y una tabla donde se visualizan los vencimientos que se van registrando. Esta interfaz tiene varios vínculos a la interfaz UI_RegistrarVencimiento, que incluye un formulario con los datos necesarios para el registro del vencimiento de principal, interés ordinario o interés moratorio según haya seleccionado el usuario. Una vez escogidos todos los vencimientos, la relación de la interfaz principal con la clase controladora CC_ControladorVencimiento produce el registro de los mismos. Esta clase contiene todas las funcionalidades necesarias para llevar a cabo este proceso, encargándose tanto de persistir los asientos contables representados por la entidad CE_AsientoVencido, como de enviar una respuesta que debe ser visualizada por el Usuario a través de la interfaz principal. De manera similar se comporta el resto de los casos de uso, aunque en algunos procesos debido a su complejidad la cantidad de clases del análisis que intervienen en el diagrama aumenta.

## 2.8 Conclusiones Parciales.

El análisis y captura de los requisitos identificados para el desarrollo de la nueva fase del subsistema Vencimiento, constituyó el punto de partida para el desarrollo del análisis de la propuesta de solución. La correcta identificación de los requisitos contribuyó a una buena calidad en el análisis de la solución, lo que favorece el diseño e implementación de la misma, comprobados finalmente por dos de las técnicas de validación de requisitos.

# CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN.

## 3.1 Introducción.

Partiendo de la disciplina de análisis, se prosigue con la de diseño, que aunque constituyen disciplinas que se encuentran definidas en un mismo flujo de trabajo (Análisis y Diseño) de la Metodología de desarrollo RUP, son actividades diferentes con artefactos diferentes. Una entrada fundamental en el diseño lo constituye el modelo de análisis, proporcionando una descripción detallada de los requisitos. Esta sección incluye el modelo de diseño del subsistema, descripción de la arquitectura y el modelo de datos. Además como artefactos generados en el flujo de trabajo implementación se muestra el diagrama de componentes y estándares de codificación.

## 3.2 Descripción de la arquitectura.

Para el desarrollo de la primera fase de Quarxo se realiza un estudio de algunas arquitecturas que existen para el desarrollo de sistemas informáticos y teniendo en cuenta sus ventajas y desventajas, su éxito para la creación de otros sistemas bancarios, la complejidad del negocio en la entidad y las características del entorno de desarrollo y el de despliegue, se concluye que la mejor opción lo constituye la arquitectura de tres niveles. Ella propone que la carga se divida en tres capas: una que se encarga del acceso a la información (acceso a datos), otra que se encarga del negocio y otra para la presentación de la información al usuario (interfaz de usuario). Dicha arquitectura aplica el patrón MVC, brindándose una clara separación entre interfaz, lógica de negocio y de presentación. Esto provee facilidad para realizar las pruebas unitarias de los componentes, reutilización de los mismos y simplicidad en el mantenimiento de los sistemas. El sistema Quarxo se encuentra compuesto por subsistemas encargados de realizar los procesos afines con un área determinada, los módulos: se encargan de la gestión de los procesos específicos de un área y los componentes se encargan de brindar los servicios para los cuales fueron diseñados. El subsistema Vencimiento se encuentra compuesto por dos módulos: Gestionar Configuración en él se realizan las configuraciones para las operaciones que se ejecutan sobre los vencimientos y Gestionar Vencimiento se encarga de la gestión de los vencimientos a través de un conjunto de operaciones.

## 3.3 Modelo de diseño.

Tomando como punto de partida las especificaciones de los requisitos funcionales definidos en el Flujo de Trabajo de Requisitos, se realiza el modelo de diseño. Este incluye un conjunto de artefactos los cuales serán mostrados tomando como ejemplo el módulo Gestionar Vencimiento por contener los casos de uso más críticos.

### 3.3.1 Diagrama de Clases del Diseño.

A partir del diagrama de clases del diseño se puede describir la estructura del sistema que se desarrolla, a través de las clases, atributos y sus relaciones. A continuación se muestra el diagrama de clases del diseño desarrollado para las nuevas funcionalidades identificadas y para una mayor comprensión del diagrama, teniendo en cuenta la gran cantidad de clases identificadas en el diseño de la solución, se divide el diagrama por partes atendiendo a las capas de la arquitectura del sistema. En algunas de estas capas no se realizan variaciones en cuanto al diseño realizado en la primera fase, no obstante se muestran y se explican algunos detalles importantes para el completo entendimiento de la propuesta de solución.

**Capa de acceso a datos.**

En esta capa se encuentran las clases encargadas de la persistencia y acceso a la información de la base de datos. Las clases encargadas de tales tareas son **FinixuBaseDAO** y **FinixuAbstractBaseDAO** las cuales heredan de **BaseDAO** y **AbstractBaseDAO**, respectivamente, las cuales ofrecen un conjunto de funcionalidades básicas para persistir y acceder a la información. De esta manera las interfaces del acceso a datos solo necesitan heredar de **FinixuBaseDAO** y sus implementaciones de **FinixuAbstractBaseDAO**. A continuación se muestra un diagrama que contiene, además de estas clases, las interfaces con las declaraciones de las funcionalidades del módulo y las clases que las implementan.

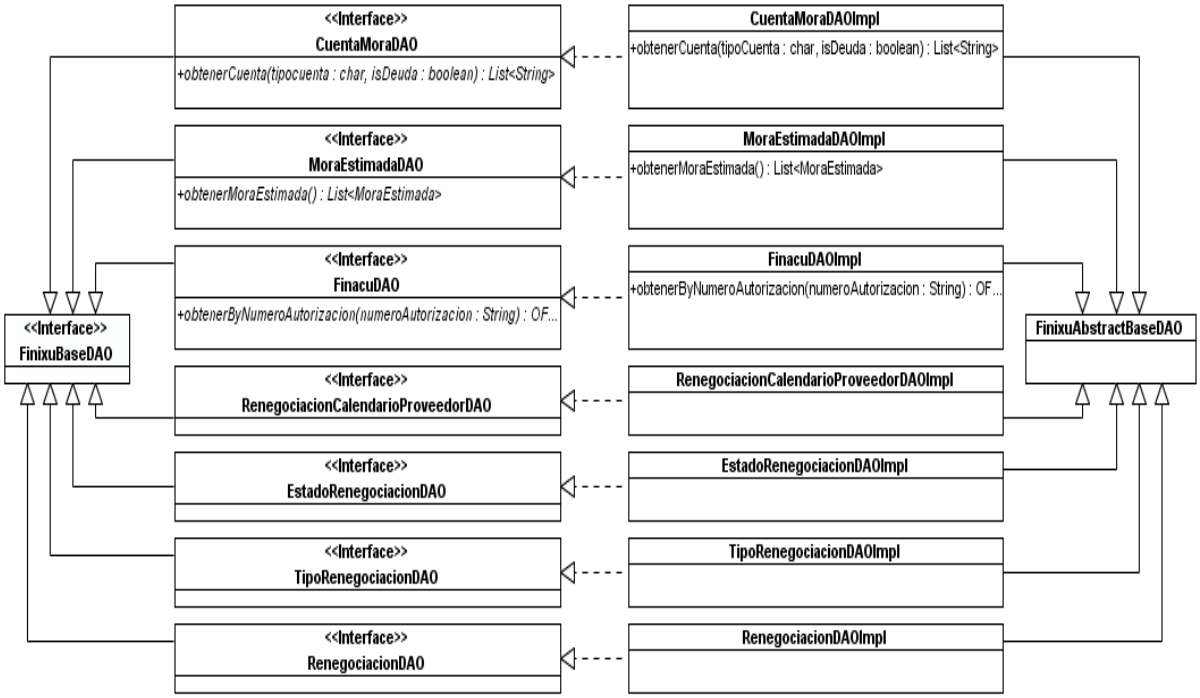


Figura 7: Diseño de la capa de acceso a datos del módulo Gestionar Vencimiento

**Capa de negocio.**

La capa de negocio cuenta con dos paquetes de clases: **manager** y **facade**. Dentro del paquete **facade**, se encuentra la clase **VencimientoFacade**, que es la interfaz encargada de contener las declaraciones de las funcionalidades que necesitará el módulo en la capa de presentación, y **VencimientoFacadeImpl**, es la encargada de implementar las funcionalidades declaradas en la interfaz **VencimientoManager**, que se encuentra dentro del paquete manager, estas funcionalidades dan solución al negocio del módulo y se encuentran implementadas por la clase **VencimientoManagerImpl**. En la siguiente imagen se muestra dichas clases y sus dependencias de manera general.

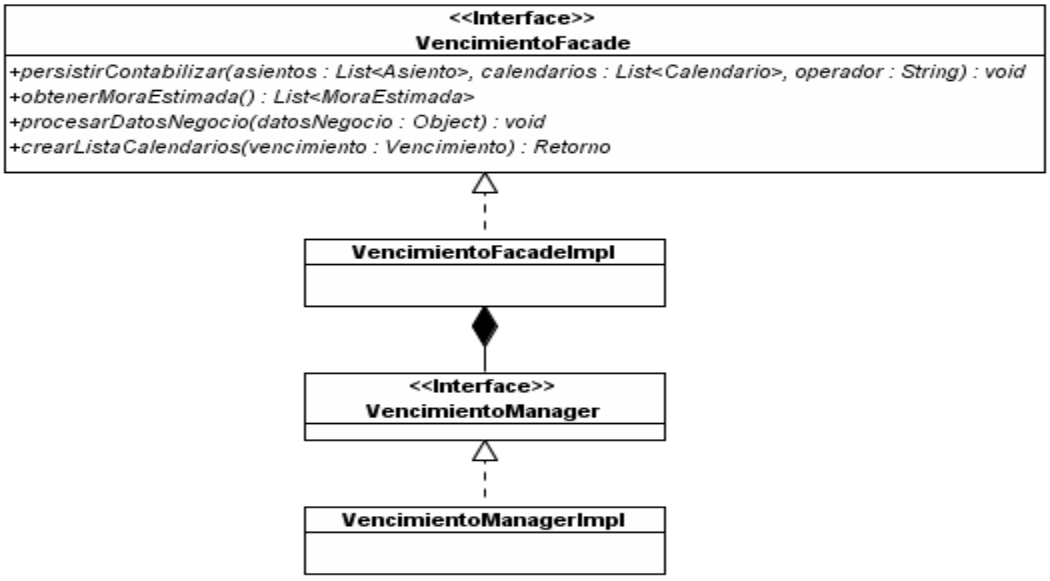


Figura 8: Diseño de la capa de negocio del módulo Gestionar Vencimiento.

**Capa de presentación.**

La capa de presentación es la encargada de mostrarle al usuario la información que este necesita. En las imágenes que se mostrarán a continuación aparecen un conjunto de clases involucradas de las cuales se dará una breve descripción de ellas para una mejor comprensión.

**ClientPage:** Constituyen páginas web encargadas de mostrar los formularios de información al usuario.

**SpringWebFlow:** Paquete de clases que gestiona las peticiones realizadas desde el cliente y mediante el cual se controlan los flujos a través de los cuales son guiados los usuarios en el sistema.

**ServerPages:** Se encargan de la creación dinámica de páginas web.

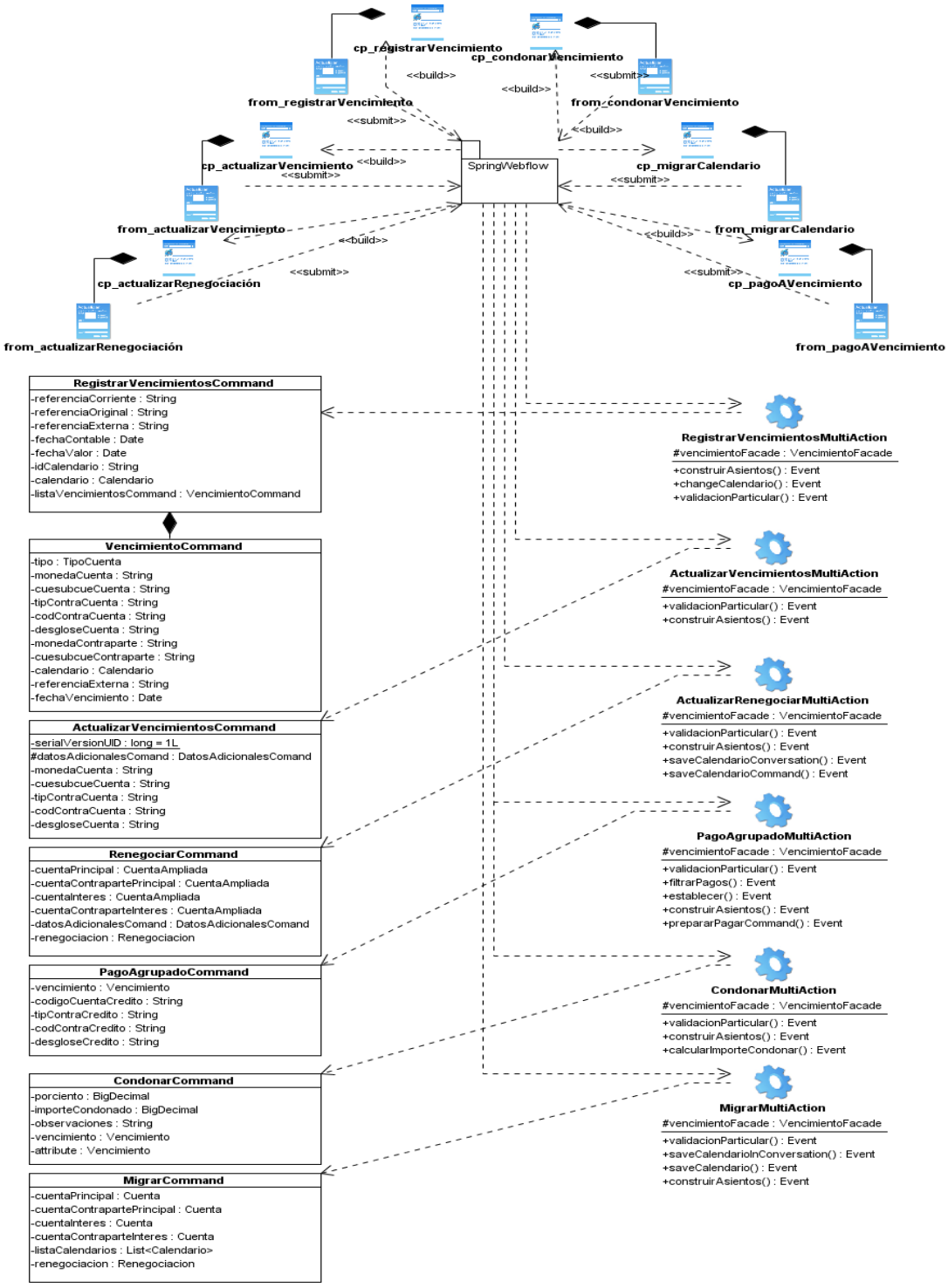


Figura 9: Diseño de la capa de presentación del módulo Gestionar Vencimiento (1).

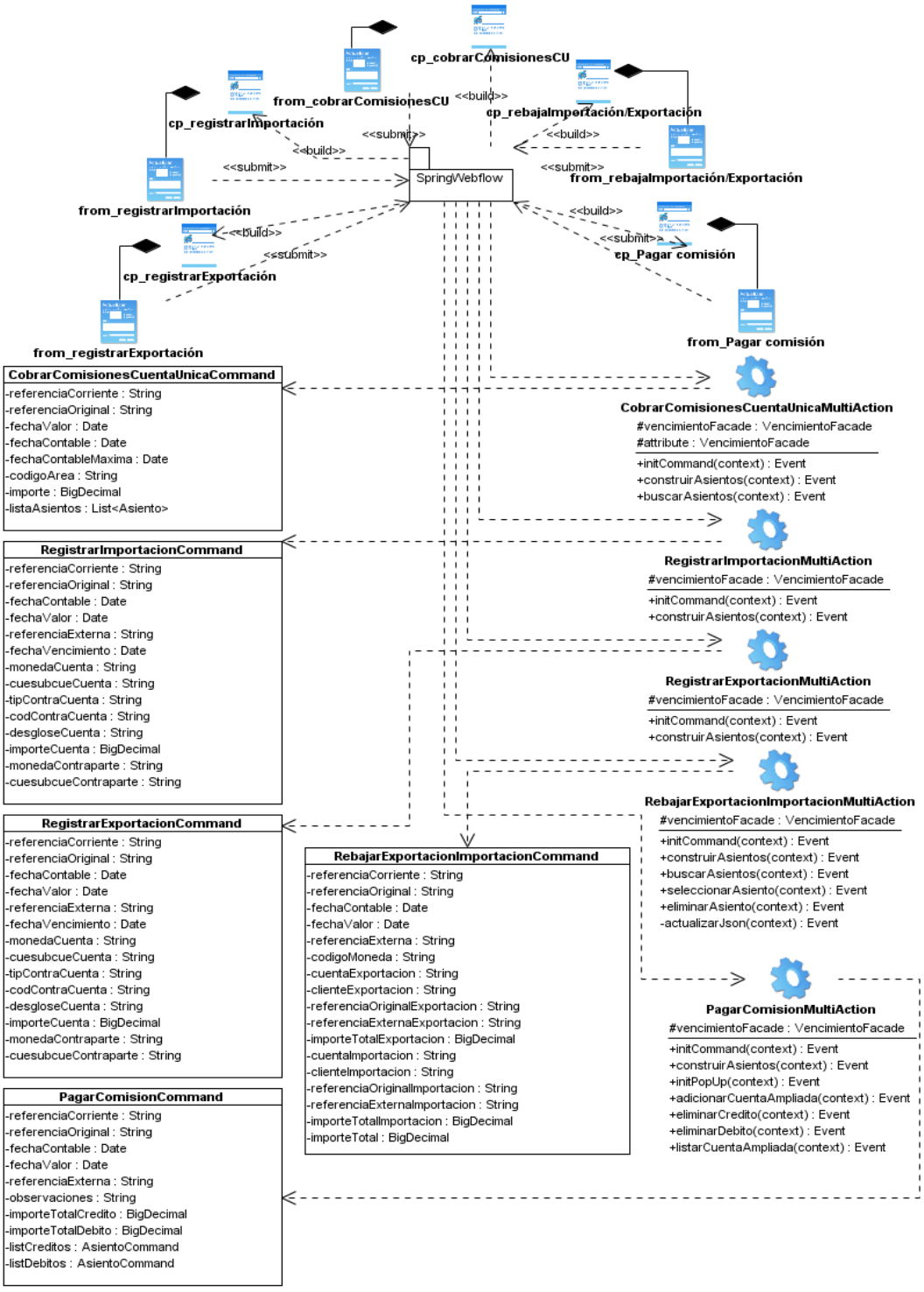


Figura 10: Diseño de la capa de presentación del módulo Gestionar Vencimiento (2).

### 3.3.2 Modelo de Paquetes

Los paquetes permiten separar un modelo en partes manejables mediante la agrupación de clases y otros paquetes. Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas, mostrando las dependencias entre ellas. A continuación se muestra el diagrama de paquetes del subsistema Vencimiento.

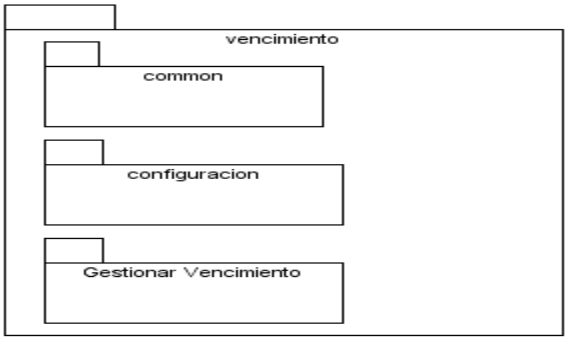


Figura 11: Modelo de paquete del subsistema Vencimiento.

Como objeto de estudio para presentar los elementos del diseño en este capítulo se selecciona el módulo Gestionar Vencimiento.

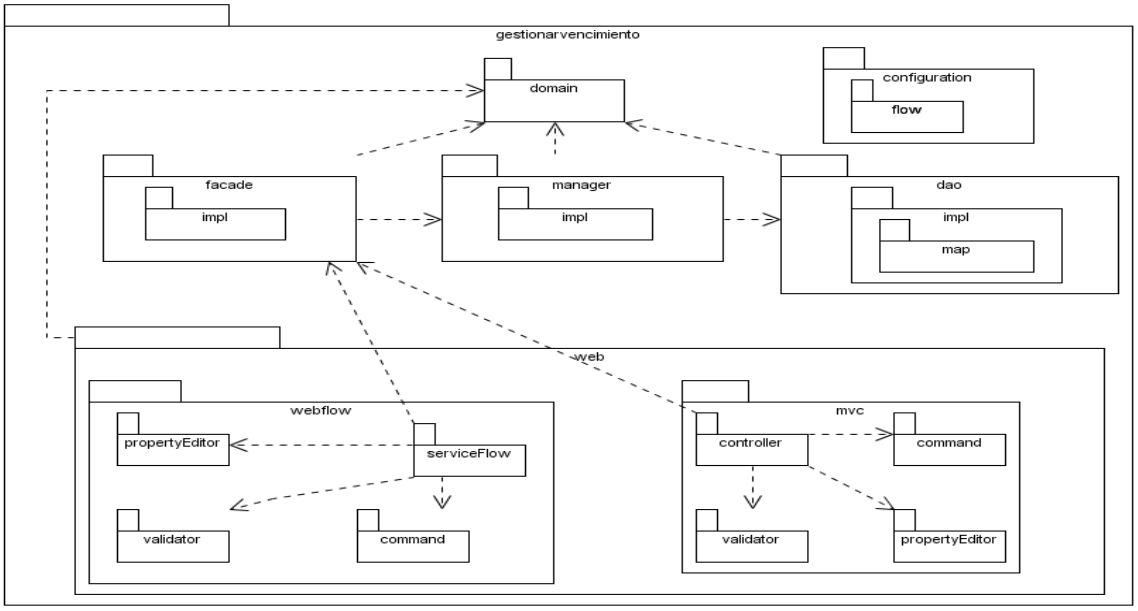


Figura 12: Modelo de paquetes del módulo Gestionar Vencimiento.

Para un mejor entendimiento del diagrama se describen a continuación algunos elementos importantes que lo componen:

**Capa de acceso a datos:**

Está compuesta por tres paquetes ***dao***, ***dao.impl*** y ***dao.impl.map***. En el ***dao*** se encuentran las interfaces DAO y StoredProceduredDAO, estas son utilizadas para el trabajo con la base de datos a través de las operaciones básicas y para invocar a los procedimientos almacenados respectivamente. En el paquete **dao.impl** están las clases que implementan las interfaces del paquete dao, y en el paquete ***dao.impl.map*** se encuentran los ficheros de Hibernate que mapean los objetos de dominio con las tablas de la base de datos.

**Capa de Negocio:**

En esta capa se encuentran las subcapas **manager** y **facade** las cuales contienen la lógica del negocio y constituyen el área de intercambio entre la capa de presentación y la de negocio respectivamente. Ambas se encuentran representadas por los paquetes **manager** y **manager.impl** en el caso de la subcapa **manager,** y **facade** y **facade.impl** en el otro caso, constituyendo **manager** y **facade** las interfaces con las declaraciones de las funcionalidades y **manager.impl** y **facade.impl** conteniendo sus implementaciones.

**Capa de presentación.**

Esta capa está compuesta por un paquete web el cual contiene dos sub-paquetes **webFlow** y **mvc**, encargándose el primero de la lógica de presentación, conteniendo dentro de sí clases con funciones específicas como:

**command:** representación de los objetos a manipular en los formularios de las páginas cliente.

**flowHandler:** personalización del trabajo con WebFlow.

**serviceFlow:** comunicación del flujo y la fachada.

**validator:** validación de los datos en el lado del servidor.

**propertyEditor:** conversión de los datos de la interfaz de usuario en objetos y viceversa.

El paquete **mvc** es el encargado de la lógica de presentación cuando se usa Spring MVC, formado por un conjunto de sub-paquetes, los cuales contiene algunas de las clases ya antes mencionadas: **validator,**

**propertyEditor y command**, además del paquete **controller**: contenedor de las clases que heredan de los controladores propuestos por este **framework**, para recibir y procesar las peticiones desde el cliente.

Además de los paquetes:

**domain:** Se localizan las clases que representan las entidades persistentes del módulo.

**configuration:** Contienen los ficheros de configuración de los diferentes contextos de Spring para el módulo:

- ✓ **servlet.xml:** contexto de Spring MVC
- ✓ **bussiness.xml:** contexto para el negocio**.**
- ✓ **webflow.xml:** contexto para SpringWebFlow.
- ✓ **dataaccess.xml:** contexto para acceso a datos.

**flow:** Contiene los archivos XML que definen los flujos de SpringWebFlow.

### 3.3.3 Diagrama de Interacción.

Los diagramas de interacción muestran la secuencia de acciones en los casos de uso y la manera en que interactúan los objetos entre sí. Existen dos tipos de diagramas de interacción: el Diagrama de Secuencia y el Diagrama de Colaboración. El primero muestra una secuencia de interacciones detalladas y ordenadas en el tiempo, y el segundo muestra una interacción organizada de los objetos que toman parte en la interacción y los enlaces entre los mismos.

Para la presentación en este capítulo se toma como objeto de estudio del módulo en cuestión el escenario registrar vencimientos.

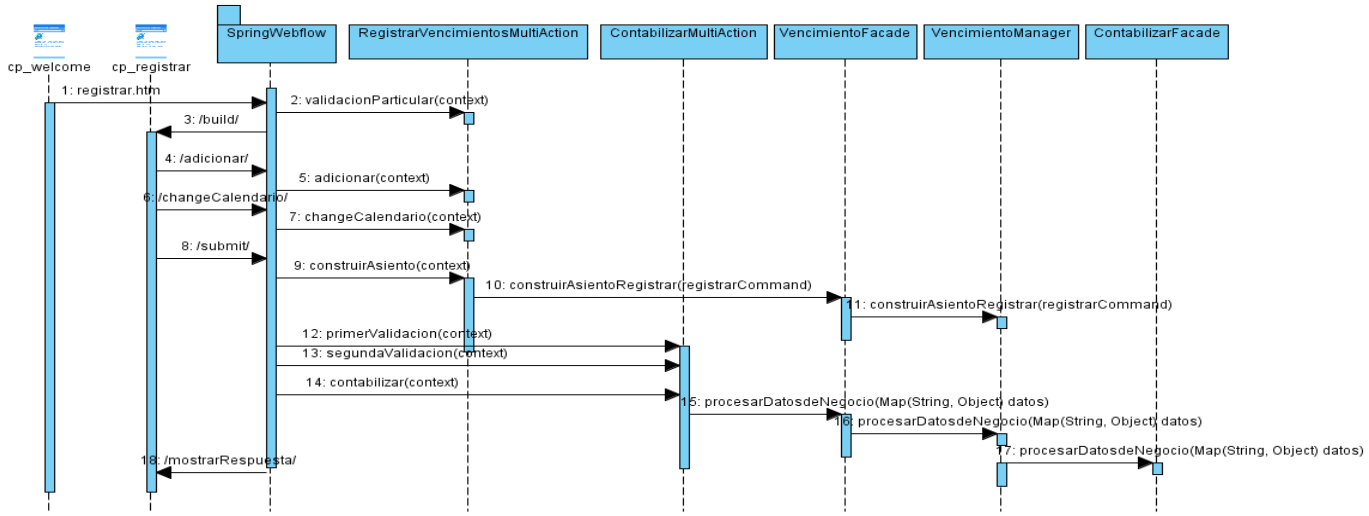


Figura 13: Diagrama de secuencia del escenario Registrar Vencimiento.

**Descripción del flujo de sucesos.**

En el diagrama de secuencia correspondiente al escenario Registrar Vencimiento, el especialista accede al mismo cuando selecciona la opción Registrar Vencimiento del menú Gestionar Vencimiento en la página cliente **welcome.jsp**. Esta realiza una petición al motor de SprinWebflow, el cual después de realizar las validaciones particulares referentes al caso de uso registrar, a través de la clase **RegistrarVencimientoMultiAction,** construye la página registrarVencimientos.jsp. La misma contiene un formulario con los campos de datos necesarios para realizar el registro del o los vencimientos que se deseen, permitiendo registrar vencimientos de interés moratorio, principal y ordinario, con sus respectivas contrapartidas, así como asociarle a los vencimientos principal y de interés ordinario un calendario existente, siendo estas acciones procesadas por la clase **RegistrarVencimientoMultiAction.** Una vez introducidos todos los datos necesarios, se pueden visualizar los asientos que posteriormente serán contabilizados, a través de una imagen en la parte derecha superior de la página, que envía una petición a la clase **ContabilizarMultiAction** de WebFlow encargada de construir los asientos y mostrarlos en una ventana de asientos contables. Además de cancelar la transacción a través del botón Cancelar, que interpreta **SpringWebFlow** como una petición de anulación de los cambios efectuados dirigiendo al usuario a la vista **welcome.jsp**. Y finalmente contabilizar que constituye el proceso crítico de la transacción. Esta petición es atendida por la clase **ContabilizarMultiAction**, la cual haciendo uso de la

fachada **VencimientoFacade** realiza las validaciones previas a las contabilización y la construcción de los asientos del registrar, muestra al usuario las advertencias en caso de que existan

## 3.4 Modelo de Datos.

El Modelo de Datos describe representaciones lógica y física de los datos persistentes utilizados por la aplicación. Permiten además, describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí. El sistema QUARXO al ser desarrollado sobre el núcleo de SABIC, fue necesario agregar algunas tablas para garantizar la persistencia de todas las entidades implicadas en la gestión de los vencimientos, a continuación se muestra el modelo de datos diseñado.

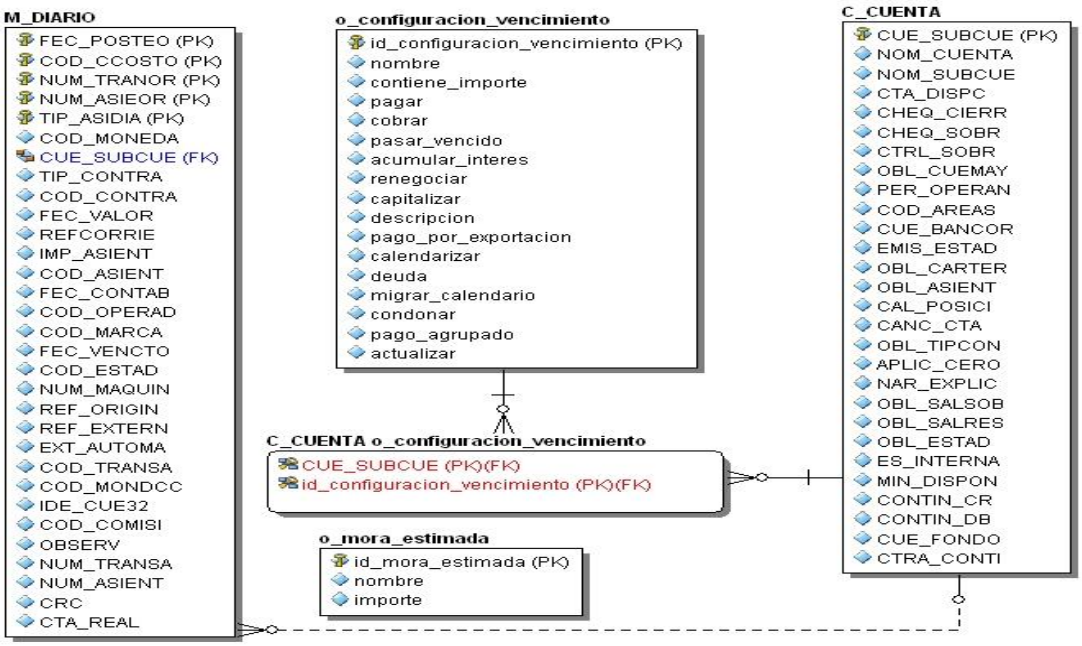


Figura 14: Modelo de Datos del subsistema Vencimiento.

## 3.5 Patrones empleados.

En el capítulo 1 se caracterizan los patrones arquitectónicos y de diseño que se aplicaron en el desarrollo del sistema Quarxo de manera general. En este epígrafe se ejemplifica el empleo de algunos de los patrones descritos en el subsistema Vencimiento. Una correcta aplicación de los patrones contribuye a que el diseño tenga la calidad y flexibilidad requerida para la implementación de la solución.

**Controlador:** La clase controladora **RegistrarVencimientosMultiAction** constituye un ejemplo de la aplicación de este patrón, centralizando las actividades que responden a las peticiones del usuario referentes al registro de los vencimientos y funcionando como intermediaria entre la lógica del proceso y la presentación.

**Experto:** Dicho patrón es evidenciado en la definición de las interfaces DAO y sus implementaciones, que se especializan en realizar las operaciones de acceso a datos referente a una determinada entidad.

**Alta cohesión:** Este patrón es utilizado en el diseño de los módulos del subsistema de manera general; donde se evidencia la agrupación de las clases en dependencia de los requisitos.

**Bajo acoplamiento:** Se emplea en el diseño de las clases del subsistema de manera general, para disminuir la dependencia entre los elementos del diseño de cada módulo, haciendo que cada uno de ellos acceda solo a las clases de nivel inferior correspondiente y se abstraiga de sus implementaciones, logrando que el código sea más fácil de entender, mantener y reutilizar.

**Fachada:** Este patrón se evidencia con la definición de la clase **VencimientoFacade** que provee a los controladores las funcionalidades del negocio del módulo, permitiendo la disminución del número de objetos con que deben trabajar y reduciendo el impacto de cambios posteriores en la lógica de negocio.

**Patrón de Acceso a Datos (DAO):** Se aplica con la definición de las interfaces DAO que disminuyen la complejidad de los objetos del dominio, al librarlos de la responsabilidad de manejar la implementación de sus fuentes de datos.

**MVC (Modelo Vista Controlador):** Se evidencia su aplicación en la arquitectura dividida en tres capas, utilizada en el proyecto SAGEB. La capa de negocio (que incluye las clases **VencimientoFacade** y **VencimientoManager**) y de acceso a datos (que contiene las interfaces DAO por cada funcionalidad y sus implementaciones) constituyen el Modelo, la capa de presentación en la parte del cliente que contiene páginas .jsp constituyen la Vista, y la capa de presentación en el lado del servidor, que engloba el conjunto de clases que hacen uso de los controladores tanto de Spring MVC como de Spring WebFlow, constituye el Controlador.

### 3.6 Validación del diseño.

Una vez desarrollado el diseño de las nuevas funcionalidades del subsistema Vencimiento se prosigue con la validación del mismo. Para efectuar la validación se utilizan las técnicas tamaño operacional de clases y relaciones entre clases; a continuación se muestran los resultados obtenidos.

**Tamaño Operacional de Clases (TOC).**

Con la utilización de esta técnica el grado de aceptabilidad y calidad del diseño es directamente proporcional al nivel de reutilización de las clases e inversamente proporcional al grado de responsabilidad y complejidad de las mismas. Como una gran cantidad de métodos en las clases afecta negativamente a los indicadores mencionados, se concluye que entre menor sea el número de métodos en las clases, mayor es el por ciento de aceptabilidad y calidad en el diseño. De manera general el diseño propuesto posee esta característica, aumentando la probabilidad de la buena calidad del mismo.

La utilización de esta técnica para validar el diseño propuesto arroja resultados positivos. Luego de realizarse las pruebas los atributos responsabilidad y complejidad de implementación reflejaron un 78% de aceptabilidad, y el atributo reutilización un 89%. Como se esperaba se evidencia que teniendo en cuenta el tamaño operacional de las clases, el diseño de la solución tiene calidad. En el anexo 1 se muestra el resultado de esta prueba.

**Relaciones entre Clases (RC).**

En esta técnica el grado de aceptabilidad y calidad del diseño es directamente proporcional a la reutilización de las clases e inversamente proporcional al acoplamiento, complejidad de mantenimiento y cantidad de pruebas de las mismas. Como la cantidad de relaciones que exista entre clases afecta negativamente a los atributos de calidad mencionados, se concluye que mientras menor sean las relaciones entre clases, mayor es el por ciento de aceptabilidad y calidad del diseño.

Con el uso de esta técnica para la validación del diseño se alcanzan resultados positivos. Luego de realizarse las pruebas los atributos acoplamiento con un 78%, complejidad de mantenimiento y cantidad de pruebas reflejaron un 89% y el atributo reutilización un 78% respectivamente. Como se esperaba se evidencia que teniendo en cuenta el tamaño operacional de las clases, el diseño de la solución tiene calidad. En el anexo 2 se muestra el resultado de esta prueba.

## 3.7 Modelo de Implementación.

En este capítulo se muestran algunos de los artefactos generados por el Flujo de Trabajo de Implementación, tales como el diagrama de componentes, los estándares de codificación y convenciones de nomenclatura.

### 3.7.1 Diagrama de Componentes.

A continuación se muestra el diagrama de componentes, donde se puede apreciar la interacción del subsistema Vencimiento y los demás componentes que se definieron en el sistema Quarxo.

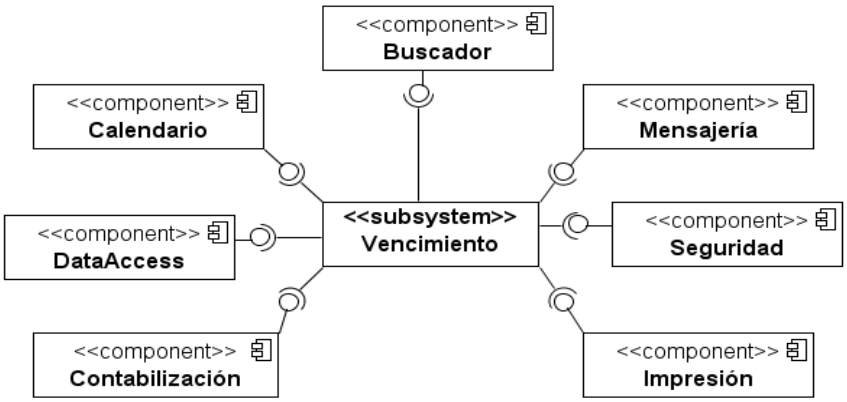


Figura 15: Diagrama de interacción de componentes.

### 3.7.2 Estándares de codificación.

Los estándares de codificación son normas que se utilizan para facilitar la lectura, comprensión y mantenimiento del código de un programa, su utilización en función de generar un código de alta calidad, permite lograr un buen rendimiento y eficiencia en el desarrollo de un software. Para del desarrollo del sistema Quarxo se utiliza la notación **PascalCasing** y **CamelCasing**. La primera se utiliza para nombrar las clases, definiendo que las mismas deben comenzar con letra mayúscula, y en caso de estar compuesta por más de una palabra, todas deben iniciar con mayúscula, ejemplo: **VencimientoManager**. En el caso de la segunda notación es utilizada para los métodos y variables que se encuentran en las clases, definiendo que deben comenzar con minúscula y en caso de contener más de una palabra deben comenzar con letra mayúscula, ejemplo: buscarDatos().

### 3.7.3 Convenciones de nomenclatura.

A partir de las notaciones definidas para la codificación de las clases del sistema Quarxo se establece una nomenclatura para las mismas según la capa de la arquitectura a la que pertenezca, facilitando su ubicación y mantenimiento del código. A continuación se muestran las nomenclaturas definidas por capas.

**Capa de acceso a datos.**

Las clases que están contenidas por el paquete dao y su sub-paquete impl presentan la siguiente nomenclatura:

Paquete dao: [Nombre de la entidad] + DAO .Ejemplo: **MoraEstimadaDAO**.

Paquete dao.impl: [Nombre de la entidad] + DAOImpl .Ejemplo: **MoraEstimadaDAOImpl**.

**Capa de negocio.**

A las clases contenidas en los paquetes facade y manager y sus sub-paquetes impl se les define la siguiente nomenclatura:

Para las interfaces: [Nombre del módulo] + [Nombre del paquete]. Ejemplo: **VencimientoManager**

Para sus implementaciones: [Nombre del módulo] + [Nombre del paquete] + Impl. Ejemplo: **VencimientoManagerImpl**.

**Capa de presentación.**

Las clases que pertenecen a los sub-paquetes mvc y webflow del paquete web tienen la siguiente nomenclatura:

Paquete controller: [Nombre de la funcionalidad a la que responde] + [Nombre del controlador de Spring MVC que se hereda]. Ejemplo: **GestionarVencimientoMultiActionController.**

Paquete serviceFlow: [Nombre de la funcionalidad a la que responde] + [Nombre del controlador de Spring WebFlow que se hereda]. Ejemplo: **BuscadorMultiAction**.

Las clases contenidas en el resto de los paquetes tienen la siguiente nomenclatura:

[Nombre de la funcionalidad a la que responde] + [Nombre del paquete]. Ejemplo: **RegistrarVencimientosCommand.**

## 3.8 Conclusiones Parciales.

El modelo de análisis constituyó el punto de partida para realizar el diseño del subsistema Vencimiento. La calidad del modelo de diseño propuesto, comprobada finalmente a través de la utilización de dos de las métricas de validación de diseño, sentó las bases para la implementación de las funcionalidades del subsistema. A partir del modelo de diseño se realiza la implementación de la segunda fase del subsistema Vencimiento.

# CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN.

## 4.1. Introducción.

Posterior al flujo de Trabajo de Implementación definido en la metodología RUP, se encuentra el de Prueba, el cual es imprescindible para verificar si el sistema cumple con los requisitos propuestos por el cliente. Los métodos utilizados para las pruebas son: Caja Blanca y Caja Negra los cuales se adecuan al nivel de prueba Unidad.

## 4.2. Aplicación de la prueba de Caja Blanca.

Para la realizar esta prueba de calidad se utiliza el framework **JUnit** el cual se puede integrar al IDE de desarrollo Eclipse, se toma como objeto de estudio el método actualizarContraparteMora() que se encuentra en la clase **BuscarVencimientoMultiActionController**, el cual se muestra a continuación.

```
public void actualizarContraparteMora(HttpServletRequest request,HttpServletResponse response){
    String  cuentaMora = request.getParameter("cuentaMora");

    ConfigVencimiento configVencimiento =
                    vencimientoFacade.getConfigVencimientoFacade().obtenerConfigVencimiento(cuentaMora);

    JSONObject json = new JSONObject();
    JSONArray array = new JSONArray();
    if(configVencimiento != null){
        int idConfiguracion = configVencimiento.getIdConfiguracion();
        List<Cuenta> contrapartidas =
            vencimientoFacade.getConfigVencimientoFacade().listarContrapartidasbyCuenta(idConfiguracion);
        if(contrapartidas != null){
            for (Cuenta cuenta : contrapartidas) {
                JSONObject data = new JSONObject();
                data.put("innerHTML", cuenta.getCueSubcue());
                data.put("value", cuenta.getCueSubcue());
                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 16: Método testeado por el framework JUnit.

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

```
public class TestJUnit extends TestCase {

    BuscarVencimientoMultiActionController buscarVencimientoMultiActionController;
    @SuppressWarnings("deprecation")
    private MockControl<HttpServletRequest> controlHttpServlet;
    private HttpServletRequest mockhttpServletRequest;
    @SuppressWarnings("deprecation")
    private MockControl<HttpServletResponse> controlHttpResponse;
    private HttpServletResponse mockhttpServletResponse;
    @SuppressWarnings("deprecation")

    protected void setUp()throws Exception {
        buscarVencimientoMultiActionController = new BuscarVencimientoMultiActionController();

        ApplicationContext context = new ClassPathXmlApplicationContext(
        "classpath:cu/uci/finixubnc/JUnit/vencimiento-context.xml");

        VencimientoManagerImpl manager = (VencimientoManagerImpl) context.getBean("vencimientoManager");
        VencimientoFacadeImpl facade = new VencimientoFacadeImpl();
        facade.setVencimientoManager(manager);

        buscarVencimientoMultiActionController.setVencimientoFacade(facade);

        controlHttpServlet= MockControl.createControl(HttpServletRequest.class);
        mockhttpServletRequest=(HttpServletRequest) controlHttpServlet.getMock();

        controlHttpResponse= MockControl.createControl(HttpServletResponse.class);
        mockhttpServletResponse=(HttpServletResponse) controlHttpResponse.getMock();

        super.setUp();
    }
```

Figura 17: Clase TestJUnit para realizar la prueba.

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

```
    @SuppressWarnings("deprecation")
    protected void tearDown() {
        controlHttpServlet.verify();
    }

    @SuppressWarnings("deprecation")
    public void testActualizarContraparteMora() throws Exception{
        mockhttpServletRequest.getParameter("cuentaMora");
        controlHttpServlet.setReturnValue("4040");
        controlHttpServlet.replay();
        assertTrue(buscarVencimientoMultiActionController.
                actualizarContraparteMora(mockhttpServletRequest,mockhttpServletResponse ));
    }
```

Figura 18: Métodos tearDown y testActualizarContraparteMora de la clase TestJUnit.

En el método tearDown es donde se verifican si las llamadas a los métodos se realizaron correctamente, es el encargado de informar la existencia de los errores en la realización de las pruebas. Las

verificaciones en este método se realizan de forma automática para todos los test definidos en la clase que se encuentra en este caso **TestJUnit**.

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

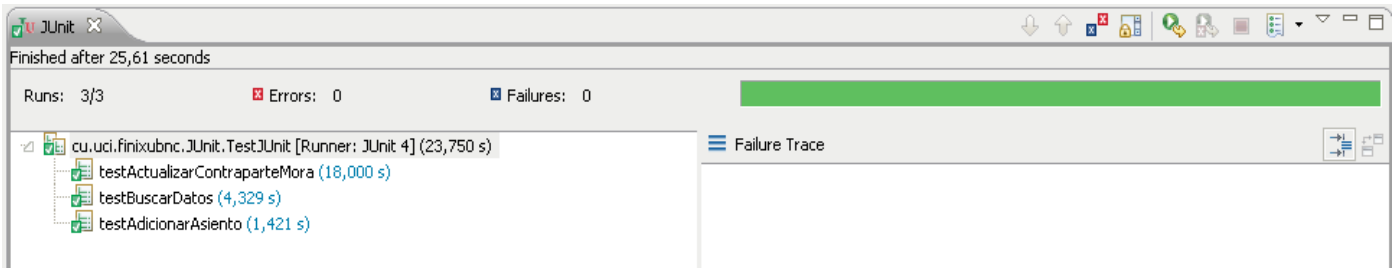


Figura 19. Resultado de las pruebas con JUnit.

### 4.3. Aplicación de la prueba de Caja Negra.

Para confeccionar los casos de prueba de Caja Negra existen diversos criterios: Particiones de Equivalencia, Análisis de Valores Límite, Métodos Basados en Grafos, Pruebas de Comparación y Análisis Causa-Efecto. La técnica empleada fue la de Particiones de Equivalencia, siendo una de las más efectivas al examinar los valores válidos e inválidos de las entradas existentes en el subsistema, permitiendo descubrir de forma inmediata un error, que de otro modo requerirían la ejecución de muchos casos de prueba antes de ser detectado. En el anexo 3 se muestra la técnica de Partición de Equivalencia tomando como caso de estudio el caso de uso Registrar Vencimiento.

### 4.4. Validación de las variables de la investigación.

La solución se somete a pruebas en un ambiente real para comprobar si se han agilizado y facilitado los nuevos procesos. Como se había mencionado al inicio del documento estos procesos solo se pueden realizar a través de la TGC de Quarxo en su primera fase. La solución a los problemas que esto conlleva lo constituye la inclusión de estas funcionalidades en el subsistema Vencimiento que respondan a estos

procesos. En términos de resultados finales los mismos no varían independientemente de la utilización de la TGC o de Vencimiento. De esta manera, el perfeccionamiento de las operaciones con el desarrollo de la solución, lo determina la agilidad y facilidad con que pueden ser realizadas las tareas por cada una de estas transacciones para obtener los resultados deseados, ocasionando una reducción del tiempo y complejidad en la ejecución. Como caso de estudio en la validación se seleccionan las funcionalidades de condonar y pago agrupado de vencimientos.

Para realizar la funcionalidad condonar vencimiento por la TGC es necesario seleccionar cada uno de los asientos a extraer, que por lo general son muchos. Luego se deben insertar las diferencias con respecto al por ciento o a la cantidad que se condona, especificando los datos de cada asiento, entre los que se encuentra el importe que es necesario calcular manualmente. Este procedimiento no solo debe realizarse para las cuentas sino también para sus contrapartidas. Para la realización de esa operación por el subsistema Vencimiento, primeramente se seleccionan las extracciones a través de un buscador que brinda muchas facilidades e información al usuario, entre las que se encuentra la opción de seleccionar un total de 50 asientos al mismo tiempo por cada página, así como visualizar los totales de los vencimientos de principal, los de interés ordinario y los moratorios, además de la cantidad de asientos. Ya dentro de la funcionalidad solo es necesario especificar el por ciento o el importe que se desea condonar. Para más información al usuario en caso de que haya especificado el por ciento, se le muestra el importe que representa del total, o si el dato introducido es el importe se visualiza el por ciento que representa. El sistema calcula automáticamente las diferencias, realiza las extracciones y las inserciones tanto de las cuentas como de sus contrapartidas. Esto demuestra la disminución de la cantidad de pasos y la complejidad en la ejecución del proceso haciendo uso de la transacción condonar del subsistema Vencimiento con respecto a la TGC.

La operación del pago agrupado puede implicar gran cantidad de asientos, cálculos de mora, liquidación de un por ciento o de un importe determinado, así como envío de mensajería SLBTR y/o SWIFT. Haciendo uso de la TGC, tanto los cálculos como la cantidad de extracciones e inserciones se realizan de igual modo que en la funcionalidad del condonar, trayendo consigo los mismos inconvenientes. Esta transacción por concepto no debe ni se encuentra integrada con los subsistemas de mensajería, ocasionando que para el envío de los mensajes SWIFT y SLBTR sea necesario acceder a dichos subsistemas por separado e introducir todos los datos correspondientes. Esto puede ocasionar inconsistencia de datos con respecto a la contabilidad y más esfuerzo debido a la cantidad de pasos en el

proceso. Como se había mencionado en Vencimiento la selección de los asientos a través del buscador es más rápida, y se incluye además cálculos automáticos de mora, y de liquidación de un por ciento o una cantidad determinada del importe de cada vencimiento. La funcionalidad del pago agrupado del subsistema, permite la generación automática del mensaje SLBTR en caso requerido exigiendo la especificación de pocos datos. Para el caso de los SWIFT se encuentra integrado automáticamente con el subsistema de mensajería, permitiendo de manera opcional la conformación y envío del mensaje. El tratamiento en la transacción de este proceso, para ambos mensajes proporciona que sean reutilizados datos correspondientes a la contabilidad, eliminando inconsistencias y la responsabilidad al usuario de introducirlos. De esta forma se evidencia la facilidad que proporciona la ejecución de esta funcionalidad a través del subsistema con respecto a la TGC.

Para demostrar cuánto se han agilizado los procesos tomados como caso de estudio se realizan en el BNC con ejemplos reales por la TGC y por el subsistema Vencimiento en su segunda versión, evidenciando los siguientes resultados.

| Operación | Vencimientos | Cantidad asientos | TGC | Vencimiento v2.0 |
|---|---|---|---|---|
| Condonar | 50 | 200 | 60-120 min | 5-10 min |
| Pago Agrupado-Parcial-Mora-SLBTR-SWIFT | 40 | 130 | 150 min | 15-20 min |

Tabla 4: Comparación de la ejecución de las funcionalidades por la TGC y el subsistema de Vencimiento v2.0.

Los datos mostrados en la tabla demuestran la considerable disminución de tiempo de ejecución de los procesos con el desarrollo de la segunda versión del subsistema Vencimiento.

### 4.5. Conclusiones Parciales.

Una vez desarrolladas las pruebas de caja negra y caja blanca, las cuales les dan cumplimiento a este capítulo, se concluye que las soluciones implementadas a los nuevos requisitos identificados dentro de la gestión de los vencimientos permiten llevar a cabo eficientemente desde un punto de vista funcional, todos los procesos involucrados en la gestión de los compromisos de cobro o pago en el BNC. De esta manera fueron incluidas en el subsistema Vencimiento del sistema Quarxo contribuyendo a la ejecución de las operaciones correspondientes.

## CONCLUSIONES.

Con el desarrollo del presente trabajo de diploma, se cumplieron los objetivos propuestos arribando a las siguientes conclusiones.

- ✓ La caracterización de la metodología, lenguajes, tecnologías y herramientas definidas para el desarrollo del sistema Quarxo y la valoración de los sistemas informáticos contables nacionales e internacionales que involucran el manejo de los vencimientos, constituyeron los elementos necesarios para fundamentar las bases teóricas para darle solución al problema planteado.
- ✓ La generación de los artefactos durante el desarrollo de los flujos de trabajo: Requerimiento, Análisis y Diseño, Implementación y Prueba proporcionó la obtención de las nuevas funcionalidades del subsistema Vencimiento.
- ✓ La validación del diseño y la implementación mediante la aplicación de las métricas y las pruebas de nivel de unidad respectivamente, demostró que las nuevas funcionalidades desarrolladas cumplen con los requisitos, eficiencia y calidad necesarios para incluirlos en el subsistema Vencimiento.
- ✓ La validación con el cliente de las nuevas funcionalidades desarrolladas demostró que las mismas disminuyen el tiempo de ejecución y la complejidad de los nuevos procesos dentro de la gestión de los vencimientos.

De manera general se le dio solución al problema existente a través de la realización de todas las tareas definidas, obteniéndose como resultado la nueva versión del subsistema Vencimiento, que permitirá, a través de sus funcionalidades mejorar la gestión de los vencimientos en el Banco Nacional de Cuba en cuanto a tiempo y disminución de los errores en los procesos.

RECOMENDACIONES

- ✓ Se recomienda utilizar el presente trabajo como referencia en el desarrollo de futuros sistemas que gestionen los compromisos de cobros y pagos.

## REFERENCIAS BIBLIOGRÁFICAS.

1. Fácil, B. *¿Qué es un Banco?* [cited 2011 12/11/2011]; Available from: http://www.bancafacil.cl.
2. Sibanc. *Banco Central de Cuba*. 2009 [cited 2011 13/11/2011]; Available from: http://www.bc.gov.cu/.
3. Noriega, A.M., *Contabilidad Bancaria*. 2 ed. Serie de Estudios Financieros-Contable 5, ed. 2. Vol. 5. 2003.
4. Daylen Barbán Reyes, R.R.R., *Diseño e implementación del subsistema Vencimiento del sistema Quarxo para el Banco Nacional de Cuba*. 2011, Universidad de las Ciencias Informáticas: La Habana.
5. Española, R.A. *Real Academia Española*. Available from: http://www.rae.es.
6. Venezuela, B.N.d. *ABC Económico*. 2012; Available from: http://www.bcv.org.ve.
7. Sibanc, *Manual SLBTR-Bancos*. 2012, La Habana.
8. SWIFT. *SWIFT*. 2012; Available from: http://www.swift.com.
9. Bravo, O. *Open Bravo, Opening ERP's Future*. 2007-2012; Available from: http://www.openbravo.com/es/.
10. SAP. *SAP for BANKING*. 2002-2012 [cited 2012 30/01/2012]; Available from: http://www.sap.com/argentina/index.epx.
11. ERP, O. *Open ERP*. 2012; Available from: http://www.openerp.com.
12. Sanabria, M.L.C.T.y.E.J.T., *Demendan Esfuerzo Extra de los Bancarios en el 2006*, in *Revista del Banco Central de Cuba*. 2006.
13. I. Jacobson, G.B., J. Rumbaugh, *El Proceso Unificado de Desarrollo de Software*. 2000, Madrid.
14. Carlos Mario Zapata, C.P., Natalí Olaya., *UNC-ANALISTA: Hacia la captura de un corpus de requisitos a partir de la aplicación del experimento mago de OZ,Escuela de Ingeniería de Antioquia*. Revista EIA ed. 2007, Medellín (Colombia).
15. Sommerville, I., *Ingeniería de Software*. 7 ed, ed. 7. 2005, Madrid: Editorial Pearson Education, S.A.
16. Pressman, R.S., *Ingeniería del Software: Un enfoque práctico*. McGraw-Hill ed, ed. 5. 2005, Madrid.
17. Informáticos, D.d.L.y.S. (1993-2012) *Patrón Arquitectónico*. Patrón Arquitectónico.
18. Jurado, I.J.L. *Facultad de Ingeniería Electrónica y Telecomunicaciones*. Universidad de Cauca.
19. Larman, C., *UML y Patrones*. Segunda Edición ed. 1999: Prentice Hall.

20. Informáticos, D.d.L.y.S. (1993-2012) *Diseño de la Capa de datos*. Patrón DAO.

21. Javier Recuenco Calvo, P.R.F. (2005) *El Patrón Composite*.

22. Ciberaula. *Ciberaula*. 2010 [cited 2012 2/3/2012]; Available from: http://java.ciberaula.com/articulo/que_es_java.

23. Morera, R.H., *Componente para la configuración visual de los servicios de integración en el marco de trabajo Sauxe.* 2011, UCI: La Habana

24. Escofet, C.M., *El lenguaje SQL*. 2008.

25. Chritian Bauer, G.K., *Hibernate in Action*. 2005, Greenwich.

26. Java 2 Platform, E.E.J.E.O. *Oracle Sun Developer Network*. 2010 [cited 2012 15/3/2012]; Available from: http://java.sun.com/j2ee/overview.html.

27. Tomcat, A. *Apache Tomcat*. [cited 2011 7/10/2011]; Available from: http://tomcat.apache.org/.

28. Microsystems., S. *Eclipse*. [cited 2012 12/3/2012]; Available from: http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int_eclipse.htm.

29. Paradigm, C.V. *Visual Paradigm*. [cited 2011 5/12/2011]; Available from: http://www.visual-paradigm.com.

30. TIC., G.d.S. *Punto de Encuentro para Proveedores y Compradores Tecnológicos*. 2011 [cited 2011 2/12/2011]; Available from: http://www.guiadesolucionestic.com/software-del-sistema/herramientas-de-desarrollo/herramientas-de-desarrollo-de-software/972-embarcadero-erstudio.

31. Subversion, C.d.v.c. *Control de versiones con Subversion*. 2011 [cited 2011 10/12/2011]; Available from: http://svnbook.red-bean.com.

32. Server, M.S. *Microsoft SQL Server*. 2006 [cited 2012 15/2/2012]; Available from: http://www.microsoft.com/spain/sql/productinfo/overview/what-is-sql-server.mspx.

33. Milián, V. *Universidad de San Buenaventura*. 2010 [cited 2011 1/3/2011]; Available from: http://usbvirtual.usbcali.edu.co/ijpm/images/stories/documentos/v1n2/003.pdf. .

34. Craig Walls, R.B., *Spring in Action Second Edition*. 2007, Estados Unidos: Manning Publications.

35. Rusell, M.A., *Dojo, The Definitive Guide*. 2008, Estados Unidos de América.

36. Massol, V., *JUnit in Action*. 2004, Greenwich: Manning Publications Co.

## ANEXOS.

### Anexo # 1. Validación del diseño aplicando la métrica Tamaño Operacional de Clase.

| | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | < =Prom. |
| | Media | Entre Prom. y 2* Pom. |
| | Alta | > 2* Prom. |

| | Categoría | Criterio |
|---|---|---|
| Complejidad implementación | Baja | < =Prom. |
| | Media | Entre Prom. y 2* Pom. |
| | Alta | > 2* Prom. |

| | | |
|---|---|---|
| Reutilización | Baja | > 2*Prom. |
| | Media | Entre Prom. y 2* Pom. |
| | Alta | <= Prom. |

Tabla 5: Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC.

| No | Subsistema | Clase | Cant. Proced. | Resp. | Comp. | Reut. |
|---|---|---|---|---|---|---|
| 1 | Vencimientos | RegistrarVencimientosMultiAction | 7 | Media | Media | Media |
| 2 | Vencimientos | ActualizarVencimientosMultiAction | 2 | Baja | Baja | Alta |
| 3 | Vencimientos | ActualizarRenegociarMultiAction | 10 | Media | Media | Media |
| 4 | Vencimientos | MigrarMultiAction | 9 | Media | Media | Media |
| 5 | Vencimientos | PagoAgrupadoMultiAction | 8 | Media | Media | Media |
| 6 | Vencimientos | CondonarMultiAction | 3 | Baja | Baja | Alta |
| 7 | Vencimientos | CobrarComisionesCuentaUnicaMultiAction | 3 | Baja | Baja | Alta |
| 8 | Vencimientos | RebajaPorExportacionMultiAction | 2 | Baja | Baja | Alta |
| 9 | Vencimientos | BuscadorMultiAction | 6 | Media | Media | Media |
| 10 | Vencimientos | ContabilizarMultiAction | 18 | Alta | Alta | Baja |
| 11 | Vencimientos | ActualizarVencimientosCommand | 0 | Baja | Baja | Alta |
| 12 | Vencimientos | CobrarComisionesCuentaUnicaCommand | 0 | Baja | Baja | Alta |
| 13 | Vencimientos | MigrarCommand | 0 | Baja | Baja | Alta |
| 14 | Vencimientos | PagoAgrupadoCommand | 0 | Baja | Baja | Alta |
| 15 | Vencimientos | RebajaPorExportacionCommand | 0 | Baja | Baja | Alta |
| 16 | Vencimientos | RegistrarVencimientosCommand | 0 | Baja | Baja | Alta |
| 17 | Vencimientos | VencimientoCommand | 0 | Baja | Baja | Alta |
| 18 | Vencimientos | BuscarVencimientoMultiActionController | 20 | Alta | Alta | Baja |
| 19 | Vencimientos | GestionarVencimientoMultiActionController | 20 | Alta | Alta | Baja |
| 20 | Vencimientos | VencimientoManagerImpl | 73 | Alta | Alta | Baja |
| 21 | Vencimientos | VencimientoFacadeImpl | 40 | Alta | Alta | Baja |
| 22 | Vencimientos | CuentaMoraDAO | 2 | Baja | Baja | Alta |
| 23 | Vencimientos | FinacuDAO | 1 | Baja | Baja | Alta |
| 24 | Vencimientos | MoraEstimadaDAO | 2 | Baja | Baja | Alta |

| | | | | | | |
|---|---|---|---|---|---|---|
| 25 | Vencimientos | AsientoVencido | 0 | Baja | Baja | Alta |
| 26 | Vencimientos | MoraEstimada | 0 | Baja | Baja | Alta |
| 27 | Vencimientos | Vencimiento | 0 | Baja | Baja | Alta |
| 28 | Vencimientos | RebajarExportacionImportacionMultiAction | 6 | Baja | Baja | Alta |
| 29 | Vencimientos | PagarComisionMultiAction | 7 | Baja | Baja | Alta |
| 30 | Vencimientos | PagarComisionMultiAction | 2 | Baja | Baja | Alta |
| 31 | Vencimientos | RegistrarImportacionMultiAction | 2 | Baja | Baja | Alta |

Tabla 6: Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad.

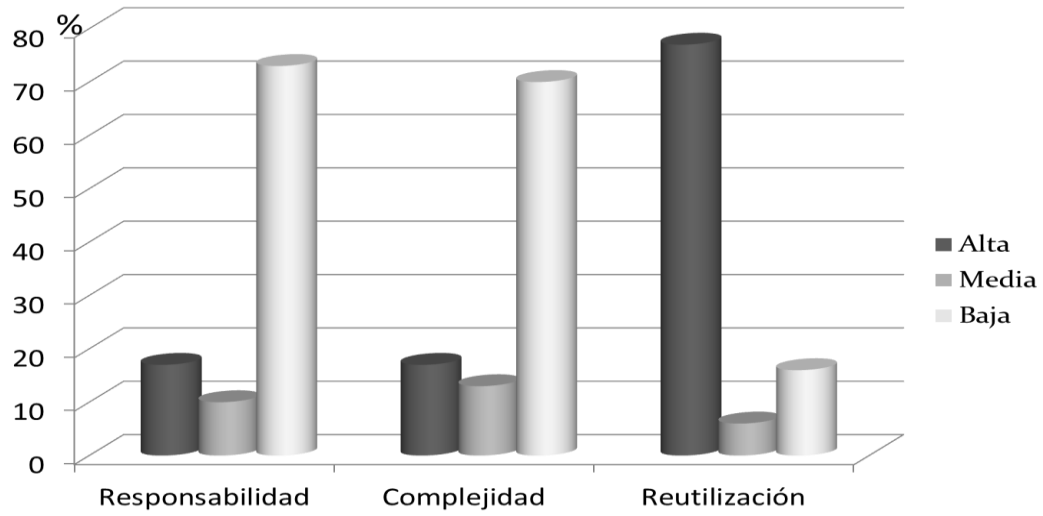


Figura 20: Resultados de los atributos de calidad evaluados en la métrica TOC.

## Anexo # 2. Validación del diseño aplicando la métrica Relaciones entre Clases.

| | Categoría | Criterio |
|---|---|---|
| Acoplamiento | Ninguno | 0 |
| | Bajo | 1 |
| | Medio | 2 |
| | Alto | >2 |

| | Categoría | Criterio |
|---|---|---|
| Complejidad de Mant. | Baja | <= Prom. |
| | Media | Entre Prom. y 2*Prom. |
| | Alta | > 2*Prom. |

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

| | Categoría | Criterio |
|---|---|---|
| Reutilización | Baja | >2* Prom. |
| | Media | Entre Prom. y 2*Prom. |
| | Alta | <= Prom. |

Tabla 7: Rango de valores de para la evaluación técnica de los atributos de calidad relacionados con la métrica RC.

| No | Subsistema | Clase | Cant. Relaciones | Acopl | Compl. Mant. | Reut. | Cant de Prueb. |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | Vencimientos | RegistrarVencimientosMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 2 | Vencimientos | ActualizarVencimientosMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 3 | Vencimientos | ActualizarRenegociarMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 4 | Vencimientos | MigrarMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 5 | Vencimientos | PagoAgrupadoMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 6 | Vencimientos | CondonarMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 7 | Vencimientos | CobrarComisionesCuentaUnicaMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 8 | Vencimientos | RebajaPorExportacionMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 9 | Vencimientos | BuscadorMultiAction | 3 | Alto | Media | Baja | Alta |
| 10 | Vencimientos | ContabilizarMultiAction | 2 | Medio | Baja | Media | Media |
| 11 | Vencimientos | ActualizarVencimientosCommand | 2 | Medio | Baja | Media | Media |
| 12 | Vencimientos | CobrarComisionesCuentaUnicaCommand | 1 | Bajo | Baja | Alta | Baja |
| 13 | Vencimientos | MigrarCommand | 8 | Alto | Alta | Baja | Alta |
| 14 | Vencimientos | PagoAgrupadoCommand | 1 | Bajo | Baja | Alta | Baja |
| 15 | Vencimientos | RebajaPorExportacionCommand | 2 | Medio | Baja | Media | Media |
| 16 | Vencimientos | RegistrarVencimientosCommand | 2 | Medio | Baja | Media | Media |
| 17 | Vencimientos | VencimientoCommand | 2 | Medio | Baja | Media | Media |
| 18 | Vencimientos | BuscarVencimientoMultiActionController | 3 | Alto | Media | Baja | Alta |
| 19 | Vencimientos | GestionarVencimientoMultiActionController | 1 | Bajo | Baja | Alta | Baja |
| 20 | Vencimientos | VencimientoManagerImpl | 15 | Alto | Alta | Baja | Alta |
| 21 | Vencimientos | VencimientoFacadeImpl | 1 | Bajo | Baja | Alta | Baja |
| 22 | Vencimientos | CuentaMoraDAO | 1 | Bajo | Baja | Alta | Baja |
| 23 | Vencimientos | FinacuDAO | 1 | Bajo | Baja | Alta | Baja |
| 24 | Vencimientos | MoraEstimadaDAO | 1 | Bajo | Baja | Alta | Baja |
| 25 | Vencimientos | AsientoVencido | 1 | Bajo | Baja | Alta | Baja |
| 26 | Vencimientos | MoraEstimada | 0 | Ninguno | Baja | Alta | Baja |
| 27 | Vencimientos | Vencimiento | 6 | Alto | Alta | Baja | Alta |
| 28 | Vencimientos | RebajarExportacionImportacionMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 29 | Vencimientos | PagarComisionMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 30 | Vencimientos | RegistrarExportacionMultiAction | 1 | Bajo | Baja | Alta | Baja |
| 31 | Vencimientos | RegistrarImportacionMultiAction | 1 | Bajo | Baja | Alta | Baja |

Tabla 8: Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad.

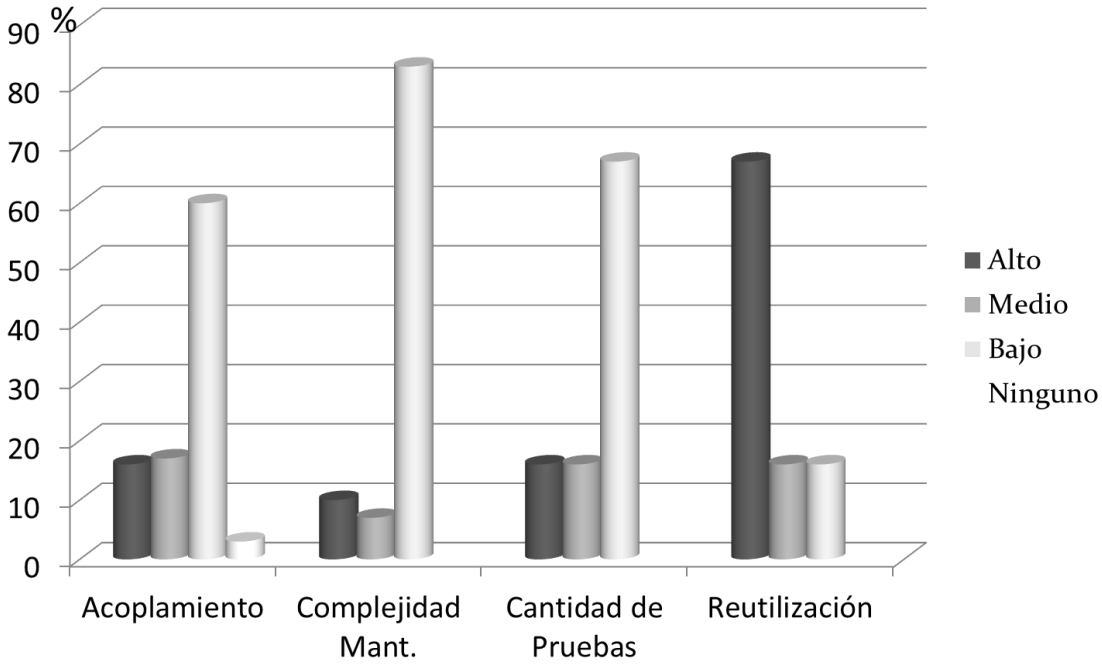


Figura 21: Figura 21: Resultados de los atributos de calidad evaluados en la métrica RC.

**Anexo # 3. Caso de prueba del caso de uso Registrar Vencimiento.**

Condiciones de Ejecución

- ✓ El usuario debe estar autenticado con los permisos necesarios para realizar la acción.
- ✓ Debe existir conexión con la Base de Datos.

| Nombre de la sección | Escenario de la sección | Descripción de la funcionalidad |
|---|---|---|
| SC 1: Registrar Vencimientos | EC 1.1: Registrar Vencimiento | El usuario llena los campos necesarios y registra el tipo de vencimiento que desea en las opciones mostradas sobre la tabla. |
| | EC 1.2: Datos incorrectos | Se verifica que los datos entrados estén correctos. En caso contrario se notifica a través de un mensaje. |
| | EC 1.3: Cancelar operación | Se cancela la operación de registrar vencimiento. |

Tabla 9: Secciones a probar en el caso de uso Registrar Vencimiento.

| No | Nombre de campo | Clasificación | Valor Nulo | Descripción |
|---|---|---|---|---|
| **1** | Referencia corriente | Campo de Texto | No | Se carga automáticamente |
| **2** | Referencia original | Campo de Texto | No | Es entrada por el usuario |
| **3** | Fecha contable | Campo de Fecha | No | Se selecciona |
| **4** | Fecha valor | Campo de Fecha | No | Se selecciona |
| **6** | Cuenta | Lista desplegable | No | Se selecciona |
| **7** | Cr | Check Box | No | Se marca si es crédito, sino será debito la cuenta |
| **8** | Importe | Campo de Texto | No | Es entrado por el usuario |
| **9** | Fecha Vencimiento | Campo de Fecha | No | Se selecciona |
| **10** | Referencia Externa | Campo de Texto | Sí | Es entrado por el usuario |
| **11** | Observaciones Asiento | Área de texto | Sí | Es entrado por el usuario |
| **12** | Calendario | Lista desplegable | Sí | Se selecciona |
| **13** | Datos adicionales | Check Box | Sí | Se marca si se desean agregarles los datos adicionales |
| **14** | Observaciones | Área de texto | Sí | Es entrado por el usuario |

Tabla 10: Descripción de las variables de entrada.

| Flujo Central de Eventos | Resp. | Ref. Corrie. | Ref. Orig | Fec. Cont. | Fec. Valor | Cuenta. | Cr. | Importe. | Fec. Venc. | Ref. Ext. | Obs. Asient. | Calen. | Dat. Adic. | Obs. | Escenario |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. El usuario introduce la URL en el navegador.<br>2. El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar.<br>3. El sistema muestra una interfaz con los subsistemas.<br>El usuario selecciona el subsistema Vencimiento.<br>4. El sistema muestra un menú con los módulos que corresponden al subsistema Vencimiento.<br>Selecciona el modulo Gestionar Vencimiento.<br>Y selecciona la opción Registrar Vencimiento.<br>5. El sistema muestra una interfaz para que el usuario realice el registro.<br>6. El usuario introduce los datos deseados y selecciona la opción Aceptar.<br>7. El sistema valida los datos, realiza el registro y muestra el mensaje : “Operación realizada satisfactoriamente” | Satisfactoria | RV1001130000 | RV1001130000 | 05/04/2012 | 04/04/2012 | 01121020000001 | true | 222 | 13/04/2012 | NA | NA | NA | NA | NA | EC 1.1: Registra Vencimiento |
| 1. El usuario introduce la URL en el navegador.<br>2. El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar.<br>3. El sistema muestra una interfaz con los subsistemas.<br>El usuario selecciona el subsistema Vencimiento.<br>4. El sistema muestra un menú con los módulos que corresponden al subsistema Vencimiento.<br>Selecciona el modulo Gestionar Vencimiento.<br>Y selecciona la opción Registrar Vencimiento.<br>5. El sistema muestra una interfaz para que el usuario realice el registro.<br>6. El usuario introduce los datos deseados y selecciona la opción Aceptar.<br>7. El sistema valida los datos, y muestra el mensaje: “No existen asientos para contabilizar | Satisfactoria | RV1001130000 | RV1001130000 | 05/04/2012 | 04/04/2012 | NA | true | 222 | 13/04/2012 | NA | NA | NA | NA | NA | EC 1.2: Datos Incorrectos |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EC1.3:Cancelar la operación | NA | NA | NA | NA | NA | 13/04/2012 | 222 | true | 01121020000001 | 04/04/2012 | 05/04/2012 | RV1001130000 | RV1001130000 | Satisfactoria | 1. El usuario introduce la URL en el navegador.<br>2. El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar.<br>3. El sistema muestra una interfaz con los subsistemas.<br>El usuario selecciona el subsistema Vencimiento.<br>4. El sistema muestra un menú con los módulos que corresponden al subsistema Vencimiento.<br>Selecciona el modulo Gestionar Vencimiento.<br>Y selecciona la opción Registrar Vencimiento.<br>5. El sistema muestra una interfaz para que el usuario realice el registro.<br>6. El usuario introduce los datos deseados y selecciona la opción Cancelar.<br>7. El sistema redirecciona a la página del subsistema. |

Tabla 11: Caso de prueba para el caso de uso Registrar Vencimiento.

## GLOSARIO DE TÉRMINOS.

**API:** Aplication Programming Interface en sus siglas en inglés, es un conjunto de funciones y procedimientos que ofrece determinada biblioteca para ser utilizada por otro software.

**Cuenta única:** cuenta en la que se deposita el dinero del estado para su posterior uso, siendo la misma controlado por el Banco Central de Cuba.

**Framework:** Es una estructura de soporte, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

**HTML:** Lenguaje para la elaboración de páginas web, es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

**IDE:** Es un entorno de desarrollo integrado, constituyendo un programa informático compuesto por un conjunto de herramientas para la programación.

**Interés Ordinario:** Impuesto monetario por el uso del principal por un tiempo determinado.

**Interés moratorio:** El interés moratorio consiste en la sanción que debe imponerse por la entrega tardía del dinero de acuerdo con lo pactado en el acto contractual en el que quedó plasmado el préstamo o negociación.

**JSP:** Es una tecnología Java para crear contenido dinámico para web en forma de documentos HTML o XML.

**Principal:** Monto que se concede inicialmente en un préstamo o negociación.

**Servlet:** Es utilizado para generar páginas web de forma dinámica a partir de parámetros de una petición de un navegador web, utilizando para este objetivo el acceso a bases de datos, flujos de trabajo y otros recursos.

**XML:** Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium permitiendo definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.