UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Título:

Desarrollo de los Módulos de Calendario, Notificaciones y fase 2 de los módulos Depósitos y Nomencladores del Sistema QUARXO

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autores:

Rosalia Cuza Tamames

Silvio Manuel Ramírez Hernández

Tutor:

Ing. Eduardo Rojas Escobar

Ciudad de la Habana, junio de 2013

"Año 54 de la Revolución"

PENSAMIENTO

"Se ha apostado fuerte por la informatización. Se ha reconocido prácticamente, no sólo en teoría, que es una industria estratégica y que su desempeño afecta, de una manera u otra a las restantes ramas económicas y sociales del país."

Dr. C. Lázaro J. Blanco Encinosa

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmamos el presente a los ___ días del mes _____ del año _____.

Rosalia Cuza Tamames

_____

Firma del Autor

Silvio Manuel Ramírez Hernández

_____

Firma del Autor

Ing. Eduardo Rojas Escobar

_____

Firma del Tutor

DEDICATORIA

Con mucho amor

A mis padres

A mi novio Victor Manuel

A mi Bebé que está por nacer

…ROSALIA CUZA TAMAMES…

A mi madre, a mi padre, a mi hermano que han sido mi meta a alcanzar y a mi novia por confiar en mí y apoyarme todo el tiempo en mis decisiones y acciones, a ellos por brindarme su amor incondicional.

…SILVIO MANUEL RAMÍREZ HERNÁNDEZ….

AGRADECIMIENTOS

DE ROSALIA

A mis padres y a mi novio Victor Manuel por ayudarme a llevar a la par la culminación de una y el inicio de dos etapas importantes en mí vida, la de estudiante y el comienzo mi vida profesional y como madre.

A mi madre y a Maritza, quienes fueron las primeras en guiarme en la realización de esta investigación; junto a Maby; y a Mailen Edith por su ayuda y además por ser una excelente profesora.

A mi compañero de tesis, Silvio, a mi tutor, Eduardo y a nuestro tutor de apoyo, Manuel y demás miembros del proyecto que de una forma u otra ayudaron con la realización del presente trabajo.

DE SILVIO

Agradezco a mi familia de forma especial a mi madre, mi padre, mi novia, a mi hermano por apoyarme y confiar en mí, gracias a ellos he llegado tan lejos.

A mi tutor Edaurdo Rojas por todo el apoyo y la ayuda que me ha brindado durante este tiempo, a Rosalia mi compañera de tesis, a Maby y Mailen que por toda la ayuda les estoy más que agradecido. A Manuel, Bello, Leo, Evelio y demás miembros del proyecto que en gran medida ayudaron en la realización de este trabajo al igual que a todos los miembros del tribunal que han dado seguimiento a la elaboración del mismo.

A mis amigos que han sabido estar ahí de una forma u otra en los momentos difíciles para darme su apoyo: Angel Rafael, Jorge Fonseca, Jorge Ricardo, Luis Carlos Guerra, Ramón Borges, Yordan Remón, Radamés Fernández, Felix Rubalcaba. A mis compañeros de grupo que han transitado a mi lado durante toda la carrera y que de una forma u otra han hecho de este largo recorrido algo para nunca olvidar.

RESUMEN

El Banco Nacional de Cuba (BNC) con el propósito de lograr la automatización de sus procesos y mejorar la comunicación con entidades bancarias internacionales y demás sucursales del país, convino con la Universidad de las Ciencias Informáticas (UCI) el desarrollo de un sistema que compensara sus propósitos, y como resultado se implementó y actualmente se encuentra en ejecución la primera fase del Sistema de Gestión Bancaria: Quarxo. Este cuenta con una serie de susbsistemas que abarcan todas las operaciones que se realizan en el BNC y que responden a los requisitos negociados en la primera fase de desarrollo del producto.

Atendiendo a la necesidad de eliminar las deficiencias detectadas en el sistema se definió una segunda fase de Quarxo para el desarrollo de nuevas funcionalidades dirigidas a mejorar los subsistemas existentes. Con tal motivo, el objetivo del presente trabajo es solucionar esas deficiencias a través del análisis, diseño, implementación y validación de nuevas funcionalidades que permitan agilizar y facilitar los procesos de creación de maquetas de calendarios, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo. En base a lo planteado a continuación se hace una caracterización de la arquitectura, patrones, lenguajes y herramientas utilizados, así como del modelo de diseño y las pruebas realizadas para la validación de la solución propuesta.

Palabras clave: Depósitos, Maquetas de Calendario, Nomencladores, Notificaciones, Quarxo.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN **10**

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA **15**

1.1 Marco conceptual 15

1.2 Estado del Arte 16

1.3 Metodología de Desarrollo de Software 19

1.4 Patrones de Diseño 21

1.5 Lenguajes de Modelado y Desarrollo 24

1.6 Herramientas de Desarrollo 25

1.7 Framework 27

1.8 Conclusiones Parciales 28

CAPÍTULO 2: REQUISITOS, ANÁLISIS Y DISEÑO **29**

2.1 Requisitos 29

2.1.1 Técnicas para la captura de los requisitos funcionales 29

2.1.2 Descripción de los requisitos funcionales y no funcionales 30

2.1.3 Validación de Requisitos 33

2.2 Análisis y Diseño 33

2.2.1 Arquitectura del Sistema 33

2.2.2 Modelo de Diseño 35

2.2.3 Validación del Diseño 44

2.3 Conclusiones Parciales 48

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS **49**

3.1 Implementación 49

3.1.1 Estándares de Codificación 49

3.1.2 Diagrama de Componentes 51

3.1.3 Diagrama de Despliegue 52

3.2 Pruebas 53

3.2.1 Pruebas Internas 53

3.2.2 Pruebas de Liberación 55

3.3 Validación de las Variables de la Investigación 62

3.4 Conclusiones Parciales 64

CONCLUSIONES GENERALES **65**

RECOMENDACIONES **66**

BIBLIOGRAFÍA **67**

GLOSARIO DE TÉRMINOS **67**

ANEXOS **71**

ÍNDICE DE FIGURAS

Figura 1: Fases del ciclo de vida de proyectos del CEIGE 19
Figura 2: Arquitectura del sistema Quarxo 34
Figura 3: Modelo de Datos del Módulo Gestionar Maquetas 36
Figura 4: Diagrama de Paquetes del Subsistema Depósitos 37
Figura 5: Diagrama de Clases del Diseño del Módulo Gestionar Nomencladores 40
Figura 6: Resultados de la métrica TOC 46
Figura 7: Resultados de la métrica RC 47
Figura 8: Diagrama de Componentes 51
Figura 9: Diagrama de Despliegue 53
Figura 10: Método a probar por el framework JUnit 54
Figura 11: Clase TestJUnit 55
Figura 12: Métodos tearDown y testEliminarMaquetaCalendario de la clase TestJUnit 56
Figura 13: Resultados de la prueba con JUnit 56
Figura 14: Prototipo no funcional Registrar Maqueta de Calendario 71
Figura 15: Prototipo no funcional Renovar Depósito a plazo 72
Figura 16: Modelo de Datos del Módulo Notificaciones 73
Figura 17: Diagrama de Paquetes del Módulo Gestionar Maquetas 73
Figura 18: Diagrama de Paquetes del Módulo Gestionar Nomencladores 74
Figura 19: Diagrama de Paquetes del Módulo Notificaciones 74
Figura 20: Diagrama de Clases del Diseño del Módulo Gestionar Maquetas 75
Figura 21: Diagrama de Clases del Diseño del Subsistema Depósitos 76
Figura 22: Diagrama de Clases del Diseño del Módulo Notificaciones 77
Figura 23: Suite de pruebas para el Módulo Gestionar Maquetas 79
Figura 24: Suite de pruebas para el Subsistema Depósitos 79
Figura 25: Suite de pruebas para el Módulo Gestionar Nomencladores 79
Figura 26: Suite de pruebas para el Módulo Notificaciones 80

ÍNDICE DE TABLAS

Tabla 1: Requisitos Funcionales del Componente Calendario 30
Tabla 2: Requisitos Funcionales del Módulo Gestionar Maquetas 30
Tabla 3: Requisitos Funcionales del Subsistema Depósitos 31
Tabla 4: Requisitos Funcionales del Módulo Gestionar Nomencladores 31
Tabla 5: Requisitos Funcionales del Módulo Notificaciones 31
Tabla 6: Requisitos No Funcionales 32
Tabla 7: Descripción de las Clases del Diseño 42
Tabla 8: Atributos de calidad que afecta el TOC 45
Tabla 9: Atributos de calidad que afecta las RC 47
Tabla 10: Escenarios de Prueba del RF Actualizar Maqueta de Calendario 59
Tabla 11: Descripción de las Variables de Entrada 60
Tabla 12: Juegos de Datos a Probar 61
Tabla 13: Tabla demostrativa para variable Facilidad 64
Tabla 14: Tabla demostrativa para variable Agilidad 64

INTRODUCCIÓN

En el mundo actual la utilización de las Tecnologías de la Información y las Comunicaciones se ha extendido, generando incontables beneficios para la sociedad. Las computadoras y las aplicaciones conocidas como software están presentes prácticamente en la mayoría de las esferas donde se maneja 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 informatizados sean cada vez más rápidos, seguros y eficientes.

En el sistema bancario cubano no existe una gran competencia, no obstante la necesidad de modernización de las aplicaciones informáticas en algunos bancos ha aumentado debido a cambios ocurridos 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, 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 un sistema informático llamado Quarxo (Sistema de Gestión Bancaria).

Quarxo posee un conjunto de subsistemas que resuelven procesos de negocio específicos que pueden estar relacionados entre sí. A pesar de eso, la primera versión del sistema Quarxo presenta deficiencias que ralentizan el trabajo dentro del BNC, por tal motivo se define una segunda fase que pretende añadir nuevas funcionalidades que permitan eliminar las actuales deficiencias. Algunos de los principales problemas son:

El sistema Quarxo no permite guardar las configuraciones de los calendarios de pago para que estas puedan ser empleadas en cualquier momento. Los operadores

actualmente, cada vez que acceden a esta funcionalidad, deben configurar el calendario de inicio a fin, ralentizando el trabajo en el BNC.

Los operadores del BNC tienen depósitos que se registran de manera repetitiva una vez llegado su plazo fin, este proceso debe hacerse de forma manual para cada uno de los depósitos debido a que el sistema no es capaz de llevar a cabo la renovación de los mismos a partir de los datos existentes.

Existen 2 nomencladores (proveedores e instituciones) que periódicamente se cargan a partir de ficheros Excel (*.xls*). El procedimiento hasta el momento consistía en eliminar todos los datos de las tablas implicadas y llenarlas nuevamente con la información de los mencionados ficheros. Ese procedimiento no es válido en el sistema actual, en tanto las tuplas de dichas tablas pudieran tener relaciones con otras de la BD. Además, los ficheros pudieran contener datos no contemplados anteriormente en la BD y que necesitan ser registrados. Por tanto la idea sería detectar conflictos de eliminación o actualización, así como adicionar los nuevos valores. Implementar transacciones para cargar la tabla de proveedores e instituciones, mostrando el resumen de elementos nuevos, eliminados y conflictos.

En cuanto a las notificaciones, el problema está dado porque en el BNC, los administradores del sistema en operación, necesitan eventualmente enviar mensajes notificando la ocurrencia de ciertos hechos (como pudiera ser el aviso de que dentro de determinado tiempo se realizará el cierre contable, o que se necesita detener la aplicación por algún motivo especial). Además, el envío de estos mensajes pudiera ocasionar el cerrado de la sesión correspondiente o incluso del navegador web al destinatario del mismo, que puede ser un usuario específico o todos. Por la frecuencia de uso de algunos mensajes, se hace necesario mantenerlos como entidades persistentes y poder gestionar su ciclo de vida, de manera que sirvan como plantillas para el envío de los mencionados avisos.

De acuerdo con el estudio de la problemática planteada se define como **problema a resolver**: ¿Cómo agilizar y facilitar los procesos de creación de calendarios, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo?

Con el fin de darle solución al problema planteado se define como **objetivo general**:

Analizar, diseñar e implementar nuevas funcionalidades que permitan la creación de maquetas (plantillas) de calendarios y su posterior uso, la renovación de depósitos de manera automática, la gestión de nomencladores y de notificaciones para agilizar y facilitar los procesos de creación de calendarios, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo.

Se define como **objeto de estudio**: la informatización de la gestión de calendarios, depósitos, nomencladores y notificaciones en los sistemas informáticos contables.

Para cumplir este objetivo se desglosan los siguientes **objetivos específicos**:

- Fundamentar la investigación mediante la elaboración del Marco Teórico.
- Realizar el análisis, diseño e implementación de las nuevas funcionalidades del sistema Quarxo.
- Validar el análisis, diseño y la implementación de las nuevas funcionalidades del sistema Quarxo.

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 calendarios, depósitos, nomencladores y notificaciones en el BNC.

La **idea a defender** en este trabajo es la siguiente: El desarrollo de los módulos Calendario y Notificaciones, así como la realización de una segunda fase para los módulos Depósitos y Nomencladores para el sistema Quarxo que permitirá agilizar y facilitar los procesos de creación de calendarios, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo.

Con el propósito de dar cumplimiento a los objetivos planteados se definen un conjunto de **tareas a cumplir**:

- Caracterización de las tecnologías, patrones y la arquitectura definida en el proyecto SAGEB.
- Valoración de los sistemas informáticos contables nacionales e internacionales que involucran la gestión de calendarios, depósitos, nomencladores y notificaciones.
- Identificación, especificación y validación de los requisitos funcionales y no funcionales identificados.
- Modelación de los nuevos procesos del módulo Calendario.
- Modelación de los nuevos procesos del módulo Depósitos.

- Modelación de los nuevos procesos del módulo Nomencladores.
- Modelación de los nuevos procesos del módulo Notificaciones.
- Modelación del diagrama de clases del diseño de las funcionalidades del módulo Calendario.
- Modelación del diagrama de clases del diseño de las nuevas funcionalidades del módulo Depósitos.
- Modelación del diagrama de clases del diseño de las nuevas funcionalidades del módulo Nomencladores.
- Modelación del diagrama de clases del diseño de las nuevas funcionalidades del módulo Notificaciones.
- Validación del diseño de las funcionalidades del módulo Calendario.
- Validación del diseño de las nuevas funcionalidades del módulo Depósitos.
- Validación del diseño de las nuevas funcionalidades del módulo Nomencladores.
- Validación del diseño de las funcionalidades del módulo Notificaciones.
- Obtención del modelo de datos de las funcionalidades del módulo Calendario.
- Obtención del modelo de datos de las nuevas funcionalidades del módulo Depósitos.
- Obtención del modelo de datos de las funcionalidades del módulo Notificaciones.
- Obtención del modelo de datos de las nuevas funcionalidades del módulo Nomencladores.
- Implementación de las funcionalidades del módulo Calendario.
- Implementación de las nuevas funcionalidades del módulo Depósitos.
- Implementación de las funcionalidades del módulo Notificaciones.
- Implementación de las nuevas funcionalidades del módulo Nomencladores.
- Validación de la implementación de las funcionalidades del módulo Depósitos, el módulo Calendario, el módulo Nomencladores y el módulo Notificaciones.
- Validar a través de pruebas de caja negra directamente con el cliente que el módulo Depósitos y el módulo Calendario facilitan y agilizan los procesos dentro de la gestión de calendario, depósitos en términos de tiempo de ejecución y complejidad en la realización de los mismos; además, que el módulo Notificaciones y el módulo Nomencladores tengan el mismo efecto sobre los procesos dentro de la gestión de nomencladores y notificaciones, en términos de tiempo de ejecución y complejidad en la realización de los mismos.

Como **posible resultado** se espera la obtención de los módulos Calendario y Notificaciones, así como la realización de una segunda fase para los módulos Depósitos y Nomencladores para el sistema Quarxo que permita agilizar y facilitar los nuevos procesos dentro de la gestión de calendarios, depósitos, nomencladores y notificaciones en el Banco Nacional de Cuba.

Estructura del Documento:

Capítulo 1: Fundamentación Teórica

En este capítulo se abordan los principales conceptos relacionados con los contenidos de Calendario, Depósitos y Nomencladores. Se describen dentro del estado del arte algunos de los sistemas de gestión bancaria existentes en la actualidad, además de la metodología y los patrones de diseño relacionados con la investigación. Incluye también la descripción de los lenguajes, herramientas y framework que serán usados en el desarrollo de la solución.

Capítulo 2: Requisitos, Análisis y Diseño

En este capítulo se presentan las fases de requisitos, análisis y diseño, explicando los requisitos funcionales por cada módulo a desarrollar y la descripción de la arquitectura del sistema Quarxo. Se incluyen los modelos y diagramas que se generan en el diseño, para el soporte de los requisitos y que dan paso a la fase de implementación.

Capítulo 3: Implementación y Pruebas

Este capítulo se centra principalmente en la implementación de los módulos que integran la solución: Calendario, Depósitos, Nomencladores y Notificaciones, la descripción de las clases y funcionalidades, así como, el modelo de componentes y la relación de los estándares de codificación utilizados. Por último, el capítulo aborda acerca de las pruebas de software que fueron realizadas al código y a la interfaz de las funcionalidades desarrolladas.

Finalmente, el documento recoge las conclusiones generales, recomendaciones y demás información anexa.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

El marco teórico orienta la dirección de una investigación exponiendo todas aquellas teorías, investigaciones y antecedentes relacionados con el tema que ayuda a la interpretación de los resultados respaldando todo el proceso de desarrollo y análisis del trabajo. A continuación el presente capítulo engloba el marco conceptual, estado del arte, lenguajes y herramientas de desarrollo relacionados con la investigación.

1.1 Marco conceptual

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. (1)

Calendario

Un calendario engloba varias formas de pago que contendrán los vencimientos, representando un acuerdo entre las partes para establecer cronogramas de compromisos de pagos. Los tipos de vencimientos que contiene pueden ser de principal o de intereses ordinarios, los cuales se pueden pagar o cobrar parcialmente, trayendo consigo que se puedan contabilizar en varios vencimientos en fechas establecidas en el convenio**.** (1)

Depósitos

Los depósitos a plazo son aquellos depósitos que sólo pueden ser retirados luego de un término determinado y con ellos se gana un interés, pero quedan inmovilizados para el depositante durante el plazo fijado. (2)

Los depósitos a plazo en BNC se realizan fundamentalmente entre bancos, no es muy frecuente el depósito a los clientes jurídicos. Estos depósitos se clasifican:

- Depósitos concedidos: cuando el BNC entrega un monto grande por el que cobra intereses y el monto total al finalizar el periodo pactado.
- Depósitos recibidos: donde BNC recibe un monto grande y luego paga intereses y monto total.
- Depósitos *Overnigth*: son depósitos que concede BNC a otros bancos pero se caracterizan por tener una corta duración: 2 o 3 días como máximo.

Nomencladores

El nomenclador o clasificador de cuentas constituye el conjunto de conceptos generales que permiten distinguir y unificar criterios, facilitar el registro de operaciones, y obtener información relevante en un período de tiempo determinado, buscando establecer una clasificación flexible, ordenada, homogénea y pormenorizada de las cuentas que se utilizan para el registro de las operaciones contables. (2)

1.2 Estado del Arte

El sistema empresarial financiero ha evolucionado cada vez más, en los últimos años, hacia la implantación de técnicas y tecnologías en el desarrollo productivo de sus instituciones.

La informatización ha significado la planificación, organización y control de los procesos de negocio. Más de una vez se ha reconocido la importancia estratégica de la informática para la sociedad, se ha apostado fuerte por la informatización. Se ha reconocido prácticamente, no sólo en teoría, que es una industria estratégica y que su desempeño afecta, de una manera u otra a las restantes ramas económicas y sociales del país. (3)

Los sistemas bancarios, tanto nacionales como internacionales, como parte de la tendencia existente que se comentaba anteriormente, han implantado sistemas para la gestión de sus procesos, a continuación se destacan algunos de ellos.

1.2.1 Sistemas de Gestión Bancaria

Sistema Automatizado para la Banca Internacional de Comercio (SABIC)

El SABIC 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. Utiliza MS-DOS como sistema de explotación, dicho sistema operativo es obsoleto y mono tarea que puede ralentizar y entorpecer el trabajo del operador. Independientemente de las ventajas que ofrece, no brinda el tratamiento requerido a la validación de los datos de entrada impidiendo que sean detectadas de forma temprana ciertas irregularidades. (4)

El SABIC contiene en sí los procesos básicos de contabilización pero deja fuera muchos procesos que se realizan en el BNC, además de que no presenta la flexibilidad necesaria para responder eficientemente a la complejidad actual de los procesos que se llevan a

cabo en dicha institución, 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. (1)

El SABIC no contempla la informatización de los Calendarios de Pago, ni cuenta con las funcionalidades de crear plantillas o maquetas de calendarios. Los operadores realizaban esta operación de forma manual a través de cálculos hechos en ficheros Excel.

El sistema cuenta con un módulo de Depósitos, pero las operaciones que se llevan a cabo en el mismo, que no incluyen la renovación de depósitos a plazo, son pobres y muchas de ellas se realizan a través de la Transacción General de Contabilidad, una vía de solución lenta y compleja.

El trabajo con Nomencladores que realiza SABIC consiste en borrar todos los datos de las tablas y llenarlas nuevamente con nuevos valores, lo que genera conflictos ya que las tuplas de dichas tablas pudieran tener relaciones con otras de la base de datos, además de que los valores nuevos puede que no estuvieran contemplados anteriormente en la base de datos y necesitan ser registrados primeramente antes de ser incluidos.

En cuanto a Notificaciones, el sistema no brida esta funcionalidad, lo que dificulta y se hace muy engorroso para el personal el aviso de la ocurrencia de ciertos eventos.

Sistema Bancario Financiero BYTE

BYTE es un conjunto de módulos independientes e integrables que permiten la automatización de todos los departamentos, según sean las necesidades y posibilidades de inversión, lo que quiere decir que para iniciar un proyecto no se requiere realizar una gran inversión, sino en forma gradual según sea el avance de su proyecto. Este sistema contempla todas las operaciones que una institución financiera realiza, tanto administrativas como operativas, y esto lo hace con sus más de 80 módulos, los cuales, integralmente, son la única opción en el mercado que le permite un superlativo grado de automatización con la más alta tecnología. (5)

Está compuesto por diferentes módulos que en su conjunto conforman una solución integral, que solventa las necesidades de los clientes. Estos módulos son:

- Contabilidad
- Consolidación de empresas

- Colocaciones
- Captaciones
- Generador de Informes Financieros
- Presupuestos
- Flujo de Caja
- Auxiliares e Integraciones de Cuentas
- Integraciones de Gastos
- Bancos y Conciliación Bancaria
- Cuentas por Pagar
- Seguridad

TEMENOS T24

TEMENOS T24 es tecnológicamente el sistema bancario más avanzado disponible hoy día. Combina la flexibilidad y la mayor funcionalidad de negocio con la arquitectura más avanzada y escalable. Brinda el poder sin precedentes para afrontar los retos del presente y las oportunidades del futuro. Es relativamente fácil seleccionar un sistema que satisfaga los requerimientos actuales y T24 es considerado una de las mejores opciones en el mercado. Consiste en la integración de módulos que cubren el rango completo de actividades bancarias, desde la banca privada hasta el manejo de tesorería corporativa, complementando las actividades y servicios con herramientas de información excepcionales. (6)

1.2.2 Valoración de los Sistemas de Gestión Bancaria

Ambos sistemas internacionales contemplan la gestión de calendario, depósitos, nomencladores y notificaciones, pero dado a que son softwares privativos no brindan la posibilidad de ver el código fuente para su estudio, impidiendo que se puedan utilizar y adaptar para el desarrollo de las nuevas funcionalidades.

En cuanto al SABIC, las operaciones que contempla (Depósitos y Nomencladores) se realizan de manera engorrosa y lenta provocando la alta ocurrencia de errores, y en conjunto con las irregularidades detectadas, hacen que este sistema no responda eficientemente a las necesidades operacionales de los procesos del BNC.

El sistema Quarxo cuenta con una serie de susbsistemas, entre ellos Vencimiento, que involucra la configuración de calendarios de pago, Depósitos y Contabilidad que contiene

la gestión de nomencladores. Dichos subsistemas responden a los requisitos pactados en la primera fase de desarrollo, lo que brinda numerosas ventajas como el uso de la arquitectura, herramientas y frameworks; que en conjunto con los requisitos acordados en la segunda fase definida, se crea el ambiente de desarrollo ideal para la implementación de las nuevas funcionalidades.

1.3 Metodología de Desarrollo de Software

El proceso de informatización de la sociedad cubana, ha propiciado el aumento del uso de herramientas informáticas en los principales sectores del país. En tal sentido, la Universidad de las Ciencias Informáticas (UCI) desempeña un rol protagónico desde su surgimiento mediante la producción de soluciones y servicios informáticos. Anexo a la Facultad 3 de la propia universidad se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE). Dicho centro productivo dirige sus resultados hacia la esfera de la gestión de empresas y entidades. La producción se concentra en el desarrollo de proyectos generalmente de gran magnitud, por lo que se hace necesario contar con un modelo estandarizado, que establezca las distintas fases por las que se debe transitar y el conjunto de artefactos a generar en cada una de ellas. El Modelo de desarrollo de software para el CEIGE detalla el ciclo de vida de sus proyectos con la incorporación de los distintos subprocesos dictados por el Nivel II de Modelo de Madurez de la Capacidad de Integración (CMMI), certificación obtenida por el centro en julio de 2011 y reconocida por el Instituto de Ingeniería de Software (SEI) como aval de la calidad de su proceso de desarrollo de software. (7)

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

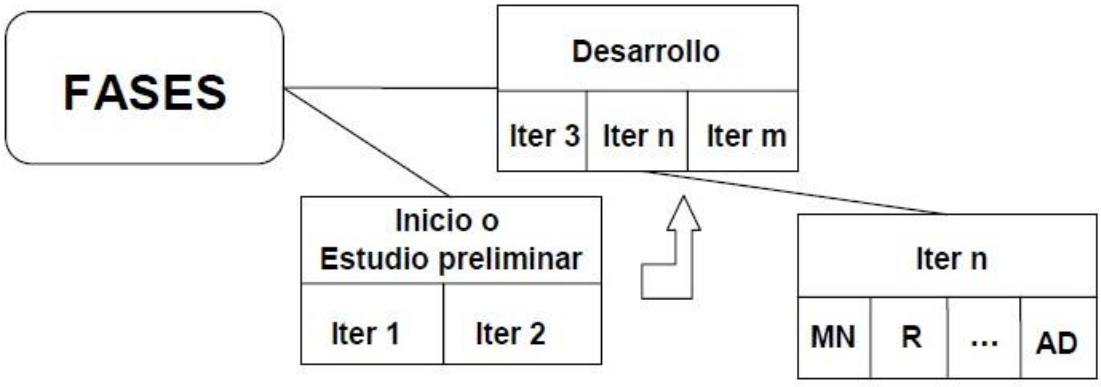


Figura 1: Fases del ciclo de vida de proyectos del CEIGE

Inicio o Estudio preliminar: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto a un alto nivel, la evaluación de la factibilidad del proyecto y el registro de este. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo, y decidir si se ejecuta o no el proyecto. Los objetivos de la fase son: (7)

- Asegurar la factibilidad del proyecto.
- Establecer un plan para la ejecución del proyecto.

Desarrollo: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. El objetivo de esta fase es obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales. En esta fase se ejecutan las disciplinas Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación. (7)

1.3.2 Descripción del ciclo de vida de los proyectos

Fase Estudio Preliminar: Se llevan a cabo las actividades relacionadas con la planeación del proyecto a un alto nivel. Se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto y realizar estimaciones de tiempo, esfuerzo y costo, y decidir si se ejecuta o no el proyecto. (7)

Disciplina Modelado del Negocio: Es la fase destinada a comprender los procesos de negocio de la organización. Se comprende cómo funciona el negocio que se desea automatizar para tener garantías de que el software desarrollado va a cumplir su propósito. (7)

Disciplina Requisitos: El esfuerzo principal en la fase de Requisitos es desarrollar un modelo del sistema que se va a construir. Incluye un conjunto de artefactos que describen todas las interacciones que tendrán los usuarios con el software y que responden a los requisitos funcionales del sistema. Se especifican los requisitos funcionales y no

funcionales. (7)

Disciplina Análisis y Diseño: Durante esta fase es modelado el sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase. Los artefactos generados en esta etapa son más formales y específicos de una implementación. En caso de llevarse a cabo la reutilización de componentes software ya desarrollados, durante esta fase se ajusta el modelado existente a los requisitos actuales. (7)

Disciplina Implementación: A partir de los resultados del análisis y diseño se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ejecutables y similares. Al reutilizar componentes software ya implementados se lleva a cabo el desarrollo necesario para ajustar a los requisitos actuales y posteriormente realizar la integración de los componentes. (7)

Disciplina Pruebas Internas: Durante esta fase se desarrollan las pruebas del grupo de calidad del centro verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software, es decir requisitos que el producto debería cumplir y que aún no los cumple. (7)

Disciplina Pruebas de Liberación: Se aplican pruebas diseñadas e implementadas por el Laboratorio Industrial de Pruebas de Software a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación. (7)

1.4 Patrones de Diseño

Un patrón de diseño es una descripción de clases y objetos que se comunican entre sí, adaptada para resolver un problema general de diseño en un contexto particular. Facilitan la localización de los objetos que formarán el sistema, así como el aprendizaje y la comunicación entre programadores y diseñadores. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). Ayudan a construir software basado en la reutilización, a construir clases reutilizables. (8)

1.4.1 Patrones de Asignación de Responsabilidades (GRASP)

Los patrones GRASP describen los principios fundamentales del Diseño Orientado a

Objetos (DOO) y la asignación de responsabilidades expresados como patrones. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. (9)

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. (1)

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 se debe conectar con el objeto producido en cualquier evento. (11)

Patrón Controlador: Un controlador es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón proporciona la llamada a los servicios más importantes de la capa de interfaz de usuario hacia las otras capas. (11)

Patrón Bajo Acoplamiento: Pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir el diseño de clases más independientes, que no se relacionen con muchas otras, que reducen el impacto de los cambios, que son más reutilizables y acrecienten la oportunidad de una mayor productividad. (10)

Patrón Alta Cohesión: Su objetivo es asignar una responsabilidad de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. (10)

1.4.2 Patrones Estructurales de Gang of Four (GoF)

Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Facade (Fachada): Facade simplifica el acceso a un conjunto de clases o interfaces. Ofrece un punto de acceso al resto de clases, si estas cambian o se sustituyen por otras solo hay que actualizar la clase Facade sin que el cambio afecte a las aplicaciones cliente. Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. (1)

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

1.4.3 Patrón de Acceso a Datos (DAO)

Este patrón cuenta con diversas fuentes de datos de tal forma que se encapsula la forma de acceder a la fuente de datos. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula. (1)

1.4.4 Patrón Modelo Vista Controlador (MVC)

Patrón que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos (Modelo, Vista y Controlador). El Modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; el Controlador es el responsable de recibir los eventos de entrada desde la Vista. (12)

- Modelo: Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.
- Vista: Este presenta el Modelo, usualmente la interfaz de usuario. La vista es la capa de la aplicación que ve el usuario en un formato adecuado para interactuar, en otras palabras, es nuestra interface gráfica.
- Controlador: El Controlador es la capa que controla todo lo que puede realizar

nuestra aplicación. Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Está compuesto por acciones que se representan con funciones en una clase.

1.5 Lenguajes de Modelado y Desarrollo

Notación para la Modelación de Procesos de Negocio (BPMN)

BPMN es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma BPMN define la notación y semántica de un Diagrama de Procesos de Negocio. (13)

Lenguaje Unificado de Modelado (UML)

UML es un lenguaje que ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagramas, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo. (14)

Java

Java es un lenguaje de programación orientado a objetos, los cuales agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. Diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. (15) Java es la plataforma ideal para las soluciones de computación en red. El código binario de Java no es un lenguaje de alto nivel, sino un verdadero código máquina de bajo nivel, viable incluso como lenguaje de entrada para un microprocesador físico. (16)

JavaScript

JavaScript es un lenguaje de programación ligero y orientado a objetos. Tiene un magnifico control sobre el navegador y el contenido de las páginas. Permite la máxima interactividad entre el usuario y la página, la verificación de los datos introducidos por el

usuario, antes de enviar el formulario al servidor, la ejecución de pequeñas cantidades de información al igual que en una base de datos y el preprocesado de información antes de enviarla al servidor. Se trata de un lenguaje interpretado puro (ni compilación, ni generación de intermedios codificados de ningún tipo) y sensible a mayúsculas, aunque algunas implementaciones ignoran en parte este último extremo. (17)

Lenguaje de Consulta Hibernate (HQL)

Hibernate utiliza un lenguaje de consulta potente HQL que se parece a SQL (*Standard Query Language*). Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. Las consultas no son sensibles a mayúsculas, a excepción de los nombres de las clases y propiedades Java. También puede asignar alias a entidades asociadas o a elementos de una colección de valores utilizando una *join*. Las consultas HQL pueden incluso retornar resultados de funciones de agregación sobre propiedades. (18)

1.6 Herramientas de Desarrollo

Visual Paradigm for UML 8.0

Para la modelación y el diseño de diagramas y prototipos no funcionales de interfaz de usuario se decide hacer uso de Visual Paradigm en su versión 8.0. La misma es una herramienta de Ingeniería de Software Asistida por Computación (CASE) y propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Visual Paradigm ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. (10)

Plataforma J2EE (en inglés: Java 2 Platform, Enterprise Edition)

J2EE (en inglés: Java 2 Platform, Enterprise Edition) representa la evolución de la plataforma de desarrollo del lado del servidor hacia una especificación más madura y sofisticada. J2EE es una plataforma para crear aplicaciones de servidor, proporciona la infraestructura necesaria para satisfacer las necesidades del lado del servidor. Proporciona un entrono gestionado para componentes y define un estándar para el desarrollo de aplicaciones empresariales multicapa. Entre sus principales ventajas se

encuentra que permite soporte de múltiples sistemas operativos, frameworks y patrones de programación que permiten responder de una forma robusta y flexible a todas las demandas de este tipo de aplicaciones. (19)

Eclipse 3.5 JEE GALILEO

Eclipse es un Entorno de Desarrollo Integrado (IDE) para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Eclipse es un marco de trabajo que está compuesto por componentes que se pueden o no incluir en dependencia de las necesidades del desarrollador, a estos complementos se les llama *plugins*. (4)

Microsoft SQL Server 2005

SQL Server es una base de datos relacional destinada a aceptar aplicaciones con arquitectura Cliente/Servidor, en el que la base de datos reside en un computador central llamado servidor y cuya información es compartida por diversos usuarios que ejecutan las aplicaciones en sus computadores locales, o clientes. Dicha arquitectura propicia una mayor integridad de los datos. SQL es un servidor de base de datos basados en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración, permite además trabajar en modo cliente-servidor y promueve la escalabilidad y seguridad de la información. (20)

Apache Tomcat

Se utilizará como servidor web Apache 2.0 pues es una tecnología gratuita de código abierto compatible con muchos Sistemas Operativos. Tiene todo el soporte que se necesita para tener páginas dinámicas. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Tiene una alta configurabilidad en la creación y gestión de registros de actividad. Apache permite la creación de ficheros de registro a medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor. (21)

Subversión

Es un sistema de control de versiones libre y de código abierto conocido habitualmente como SVN que se ha expandido en gran medida dentro de la comunidad del desarrollo de software. Maneja ficheros y directorios a través del tiempo y la información radica en un árbol de ficheros en un repositorio central. Subversión puede acceder al repositorio a través de redes lo que le permite ser usado por personas en ordenadores distintos fomentando la colaboración que deriva en la reducción del tiempo de desarrollo. Existen diferentes clientes para el Subversión ya sean programas independientes como el TortoiseSVN y el Subclipse para integrarlo con Eclipse. (10)

Mozilla Firefox 3.6

Este navegador multiplataforma es libre y es compatible con los estándares web (HTML, XML, CSS y JavaScript) que se emplearán para el desarrollo a la solución que se propone. Proporciona un entorno para los desarrolladores web en el que se puede utilizar herramientas incorporadas, para probar código JavaScript. (22)

1.7 Framework

Hibernate 3.5

Hibernate framework soporta la conexión a una gran variedad de servidores de base de datos. Permite transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, persistencia transitiva. Permite la ejecución de consultas SQL (*Standard Query Language*) y además posee un potente lenguaje de consulta de datos llamado HQL (*Hibernate Query Language*), que permite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos. (23)

Dojo 1.3

Dojo Toolkit es una fuente abierta modular de librería JavaScript destinado a facilitar el rápido desarrollo de JavaScript o de las aplicaciones basadas en AJAX y sitios web. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. (24)

Spring 2.5

Spring es un framework de código abierto. Se creó para abordar la complejidad del desarrollo de aplicaciones empresariales. Cualquier aplicación Java se puede beneficiar de Spring en términos de simplicidad, capacidad de prueba, y acoplamiento. Proporciona una potente gestión de configuración basada en JavaBeans, además de una capa genérica de abstracción para la gestión de transacciones. (25)

Spring MVC

Dentro del framework de Spring se encuentra con una parte referente al modelo vista controlador. Spring MVC se puede utilizar para crear aplicaciones web escalables y robustas. Este framework es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP y otros. Emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio. (26)

Spring WebFlow 1.9.7

Spring WebFlow está dirigido a definir y gestionar los flujos de páginas dentro de una aplicación web. Permite la definición de configuraciones de controladores autónomos, conocido como flujos, que definen un diálogo conversacional entre las peticiones de usuario y las respuestas de aplicación. Flow es un término utilizado para identificar el movimiento en una aplicación web de una página a otra. Está diseñado para manejar los estados, transiciones, y eventos. (27)

1.8 Conclusiones Parciales

A partir del análisis y el estudio realizado de los sistemas de gestión bancaria se concluye que Quarxo es la solución ideal que se ajusta a las necesidades del BNC y con el desarrollo de la segunda fase este sistema aportará una mayor facilidad y agilidad en las operaciones de dicha entidad financiera.

CAPÍTULO 2: REQUISITOS, ANÁLISIS Y DISEÑO

El presente capítulo enmarca, de la fase de Desarrollo, las Disciplinas de Requisitos, Análisis y Diseño para el desarrollo de los módulos Calendario y Notificaciones y fase 2 de los módulos Depósitos y Nomencladores del sistema Quarxo. Se describen los requisitos funcionales y no funcionales, la arquitectura base definida, el modelo de datos que recoge toda la información que transforma al sistema, el diagrama de paquetes para la organización de los módulos y el diagrama de clases para el entendimiento de las clases y paquetes relacionados.

2.1 Requisitos

El análisis de requisitos es la primera fase técnica del proceso de ingeniería de software. La tarea del análisis de requisitos es un proceso de descubrimiento, refinamiento, modelado y especificación que permite definir la función y el rendimiento del software, indica la interfaz y las restricciones que debe cumplir el software. (27)

El esfuerzo principal en la fase de Requisitos es desarrollar un modelo del sistema que se va a construir. Incluye un conjunto de artefactos que describen todas las interacciones que tendrán los usuarios con el software y que responden a los requisitos funcionales del sistema. Se especifican los requisitos funcionales y no funcionales. (7)

2.1.1 Técnicas para la captura de los requisitos funcionales

Una adecuada compresión de los requisitos contribuye al desarrollo de mejores sistemas que cumplan con las necesidades y expectativas del cliente. Para llevar a cabo este procedimiento existen diversas técnicas que guían al analista en el proceso de establecer una comunicación con el cliente y el equipo de desarrollo, de las cuales fueron utilizadas las siguientes:

- **Entrevistas:** Se realizaron a través de reuniones y encuentros efectuados con los clientes para obtener la información necesaria de los procesos relacionados con calendario, depósitos, nomencladores y notificaciones. A través de estas entrevistas se alcanzó como resultado la identificación de todos los requisitos funcionales necesarios para la satisfacción del 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.

2.1.2 Descripción de los requisitos funcionales y no funcionales

La especificación de requisitos contiene una descripción completa de las necesidades y funcionalidades del sistema que será desarrollado, así como la definición de los requisitos funcionales y no funcionales.

Los requisitos funcionales definidos previamente en las entrevistas con el cliente son:

Componente Calendario: Para este componente dentro del subsistema Vencimiento se integrarán las siguientes funcionalidades:

RF1: **Crear Maqueta de Calendario**	Consiste en guardar como maqueta de calendario los datos que el usuario registra una vez que configura un calendario.
RF2: **Cargar Maqueta de Calendario**	Consiste en construir un calendario a partir de la maqueta de calendario que el usuario seleccione.

Tabla 1: Requisitos Funcionales del Componente Calendario

Módulo Gestionar Maquetas: Este nuevo módulo a desarrollar, dentro del Subsistema Vencimiento, tiene como objetivo principal permitir guardar las configuraciones de los calendarios para que estas puedan ser utilizadas en cualquier momento. El mismo contendrá las siguientes funcionalidades:

RF3: **Registrar Maqueta de Calendario**	Consiste en registrar los datos que el usuario ingresa cuando configura una maqueta de calendario.
RF4: **Actualizar Maqueta de Calendario**	Consiste en renovar los datos de la maqueta de calendario seleccionada por el usuario.
RF5: **Buscar Maqueta de Calendario**	Consiste en buscar todas las maquetas de calendario creadas anteriormente de acuerdo a los criterios de búsqueda seleccionados.
RF6: **Eliminar Maqueta de Calendario**	Consiste en eliminar las maquetas de calendario que fueron creadas anteriormente por el usuario.

Tabla 2: Requisitos Funcionales del Módulo Gestionar Maquetas

Subsistema Depósitos: Para este subsistema se integrará la siguiente funcionalidad:

RF7: **Renovar Depósito**	Consiste en actualizar los vencimientos de los intereses del depósito seleccionado por el usuario.

Tabla 3: Requisitos Funcionales del Subsistema Depósitos

Módulo Gestionar Nomencladores: para este módulo perteneciente al Subsistema Contabilidad se integrarán las siguientes funcionalidades:

RF8: **Cargar Nomencladores**	Consiste en cargar las tablas de proveedores e instituciones dados los datos que el usuario registra del nomenclador.
RF 9: **Registrar Nomencladores**	Consiste en persistir los datos registrar los datos que el usuario ingresa cuando configura los valores de las tablas.

Tabla 4: Requisitos Funcionales del Módulo Gestionar Nomencladores

Módulo Notificaciones: para este nuevo módulo dentro del Subsistema Administración se integrarán las siguientes funcionalidades:

RF10: **Enviar Notificación**	Consiste en enviar una notificación a un grupo de usuarios o a un usuario en específico para dar aviso sobre las acciones a ejecutar en el sistema.
RF11: **Enviar Orden de Cerrado**	Consiste en enviar una notificación a un grupo de usuarios o a un usuario en específico para alertar acerca del cierre del sistema.
RF12: **Gestionar Mensajes Predeterminados**	Consiste en buscar las notificaciones predeterminadas creadas anteriormente.

Tabla 5: Requisitos Funcionales del Módulo Notificaciones

Requisitos No Funcionales:

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

Tabla 6: Requisitos No Funcionales

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.

2.1.3 Validación de Requisitos

La validación de requisitos tiene como objetivo definir que los requerimientos 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. Entre las técnicas utilizadas para validar los requisitos identificados se utilizaron las siguientes.

- **Revisiones**: 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.
- **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. (Ver Anexo1)

2.2 Análisis y Diseño

Durante esta etapa es modelado el sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase.

2.2.1 Arquitectura del Sistema

La arquitectura es la estructura jerárquica de los componentes de un programa, la manera de interactuar de estos componentes y la estructura de los datos usados por estos componentes. (27)

La arquitectura del sistema de gestión bancaria Quarxo está compuesta por tres capas: la Capa de Presentación, la Capa de Negocio y la Capa de Acceso a Datos y se utiliza una capa transversal a las otras con las clases del Dominio. Esta arquitectura fue establecida con el objetivo de separar las responsabilidades en cada una de las capas y lograr una mayor reutilización, escalabilidad y facilidad para el desarrollo del sistema.

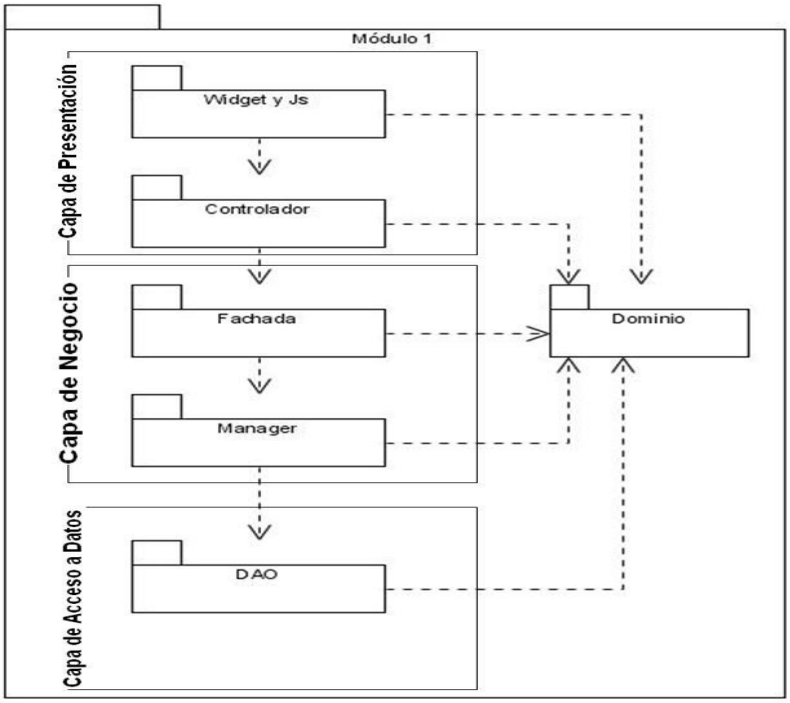


Figura 2: Arquitectura del sistema Quarxo

Capa de Presentación

En esta capa se desarrolla la lógica de presentación. En el lado del servidor para recibir, controlar y enviar una respuesta a las peticiones realizadas desde el cliente, se utilizará Spring MVC y se utilizará Spring WebFlow para representar y controlar los flujos complejos y reutilizables de la aplicación. En el lado del cliente se utilizará la librería Dojo para generar las interfaces que interactuarán con el usuario. La Capa de Presentación estará relacionada con la Capa de Negocio y la Capa de Dominio.

Capa de Negocio

Esta capa está divida en dos subcapas:

- La subcapa Fachada agrupará las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación, sin realizar la lógica de negocio, delegando de esta forma a la subcapa Manager la realización de la lógica del negocio. Esta será además, el punto de intercambio entre la capa de presentación y la capa de negocio.
- La subcapa Manager es donde se realizará la lógica del negocio conteniendo la jerarquía de clases indicadas para su implementación. Además esta subcapa

utilizará la capa de acceso a datos para el trabajo con la persistencia de los datos y la capa de dominio para la generación de los objetos del dominio.

Capa de Acceso a Datos

En esta capa se realizarán todas las operaciones relacionadas con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información de la aplicación. La interacción con la capa de negocio se realizará a través de interfaces. Con la utilización del patrón Modelo Vista Controlador (MVC) se separa la interfaz de usuario, la lógica de control y los datos de una aplicación, en tres componentes distintos: el modelo administra el comportamiento y los datos del dominio de aplicación, la vista maneja la visualización de la información y el controlador interpreta las acciones del usuario sobre el sistema, informando al modelo y/o a la vista para que cambien según resulte apropiado.

2.2.2 Modelo de Diseño

El modelo de diseño expande y detalla los modelos de análisis tomando en cuenta todas las implicaciones y restricciones técnicas. El propósito es especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente durante la fase de implementación. (28)

El diseño es el primer paso en la fase de desarrollo de cualquier producto o sistema de ingeniería. El objetivo del diseño es producir un modelo o representación de una entidad que se va a construir posteriormente. (7)

2.2.2.1 Modelo de Datos

El modelo de software debe ser capaz de modelar la información que transforma el software, las funciones que permiten que ocurran las transformaciones y el comportamiento del sistema cuando ocurren estas transformaciones. El modelo de datos se compone de tres piezas de información interrelacionadas: el objeto de datos, los atributos y la relación que conecta los objetos entre sí. (27)

A continuación se presenta el modelo de datos generado para el módulo Gestionar Maqueta y una explicación de las partes que lo conforman. (Ver demás modelos en Anexo2)

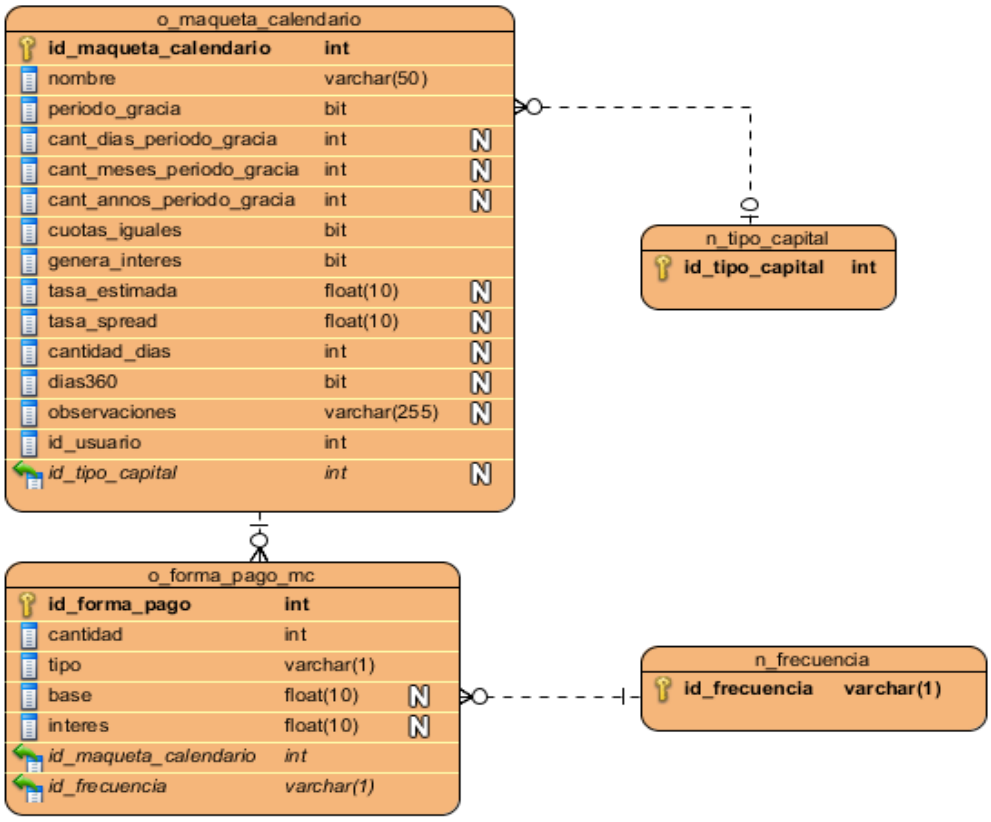


Figura 3: Modelo de Datos del Módulo Gestionar Maquetas

El modelo de datos lo conforman 4 tablas: **o_maqueta_calendario**, **n_tipo_capital**, **o_forma_pago_mc** y **n_frecuencia**. La tabla **o_maqueta_calendario** recoge los datos de las maquetas de calendario una vez que se registran en el sistema, **o_forma_pago_mc** almacena los datos relacionados con las formas de pagos que contendrán las maquetas de calendario y si las mismas generan intereses, **n_frecuencia** hace referencia a la frecuencias de pago que contendrán las maquetas de calendario, que son: al final, diario, eventual, anual, bimensual, cuatrimestral, mensual, semestral, trimestral; y **n_tipo_capital** que contiene los valores para realizar los cálculos del interés.

2.2.2.2 Diagrama de Paquetes

El diagrama de paquetes muestra como está estructurado el sistema. Cada paquete puede contener otros paquetes o clases, que tienen interfaces y realizan cierta funcionalidad. Los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. Normalmente se encuentran organizados para incrementar la coherencia interna dentro de cada paquete y reducir el acoplamiento externo entre los

paquetes. (29)

A continuación se muestra la estructura y dependencia de los paquetes que componen el subsistema Depósitos, así como, una breve descripción de cada uno de ellos. Los diagramas de paquetes que comprenden los requisitos asociados a los módulos de Gestión de Maquetas de Calendario, Nomencladores y Notificaciones se comportan de manera similar. (Ver Anexo2a)

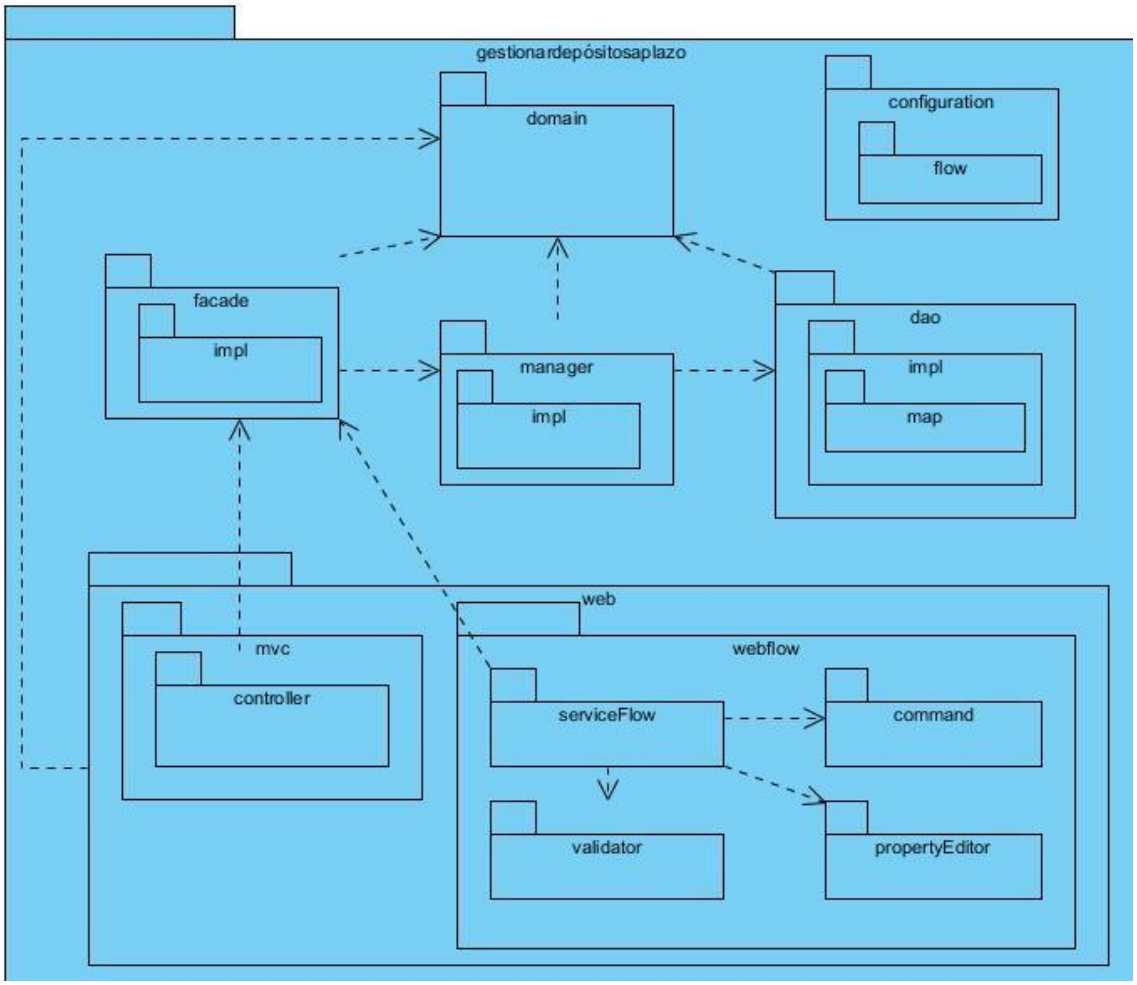


Figura 4: Diagrama de Paquetes del Subsistema Depósitos

Descripción de los paquetes:

Paquete domain: Contiene las clases del dominio del módulo.

Paquete facade: Contiene la interfaz y la implementación que brindan las funcionalidades que se consumirán por otros módulos o por la capa de presentación.

Paquete manager: Contiene las interfaces e implementaciones que brindan las funcionalidades del negocio del módulo correspondiente.

Paquete dao: Contiene las interfaces y las clases de implementación del acceso a datos del módulo, además los *maps* que son los ficheros de mapeos de hibernate.

Paquete web: Contiene un grupo de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.

Paquete mvc: Contiene los paquetes que manejan la lógica de presentación en el servidor para SpringMVC.

- **Sub-Paquete controller:** Contiene las clases que heredan de las clases *Controller* que brinda SpringMVC para responder a las peticiones realizadas por el cliente.

Paquete webflow: Se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring Web Flow.

- **Sub-Paquete serviceFlow:** Contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.
- **Sub-Paquete command:** Contiene las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.
- **Sub-Paquete validator:** Contiene las clases para desarrollar las validaciones en el servidor.
- **Sub-Paquete propertyEditor:** Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

Paquete configuration: Contiene los ficheros de configuración del módulo en formato XML de los diferentes contextos de Spring, en forma de mapeo de peticiones, controladores y vistas, los mismo son:

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

- **Sub-Paquete flow:** Se encuentran presentes los archivos XML que definen los flujos para Spring Web Flow.

2.2.2.3 Diagrama de Clases del Diseño

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema en concreto y los servicios que el

sistema debería proporcionar a sus usuarios finales. Normalmente contiene: clases, interfaces y relaciones entre ellas: de asociación, de dependencia y/o generalización. Pueden contener paquetes o subsistemas, que se usan para agrupar elementos del modelo en partes más grandes. (30)

A continuación se muestra el diagrama de clases que componen el módulo Gestionar Nomencladores y una breve descripción de cada una de ellas para su mejor entendimiento. Los diagramas de clases de los demás módulos ver en Anexo2b.

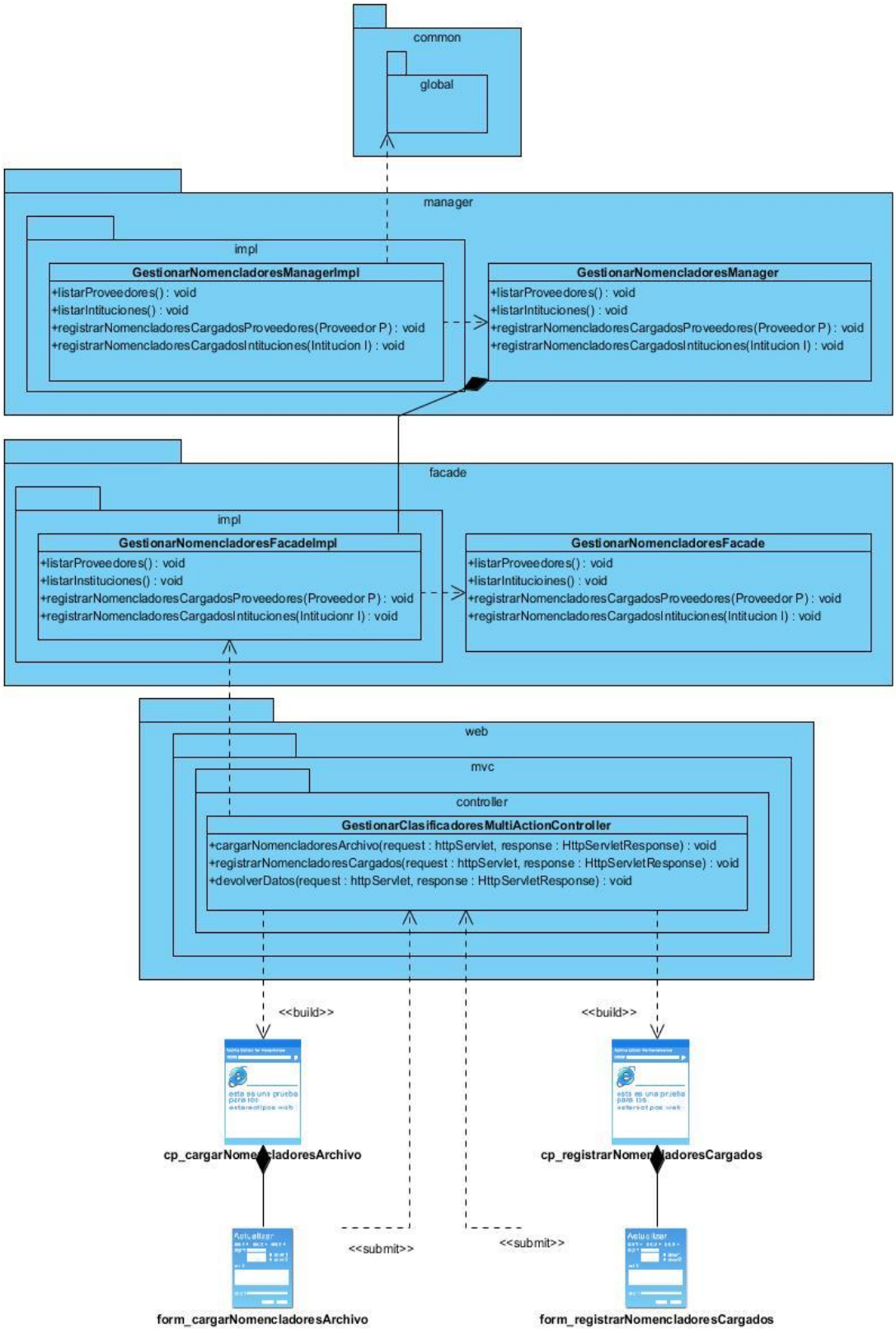


Figura 5: Diagrama de Clases del Diseño del Módulo Gestionar Nomencladores

Descripción de las clases:

Clases ClientPage (cp): Páginas web encargadas de mostrar los formularios (**form**) e información al usuario.

Clase GestionarClasificadoresMultiActionController: Permite controlar las acciones que realice el usuario con el formulario: **form_cargarNomencladoresArhivo**.

Clase GestionarNomencladoresFacade: Clase fachada del módulo Gestionar Clasificadores, a través de esta clase se puede hacer uso de todas las funcionalidades que brinda el módulo.

Clase GestionarNomencladoresManager: Clase manager del módulo Gestionar Clasificadores, es la encargada de implementar todas las funcionalidades del módulo, así como así como hacer uso de funcionalidades existentes en otros módulos y/o subsistemas.

Descripción de los atributos y métodos de las clases:

GestionarClasificadoresMultiActionController	
Atributos	
- upload :ServletFileUpload - datos: JSONObject	
Métodos	**Descripción**
# devolverDatos(HttpServletRequest request, HttpServletResponse response):void	Método encargado de escribir los datos de los nuevos nomencladores en el Response.
# registrarNomencladoresCargados(HttpServletRequest request,HttpServletResponse response):void	Método encargado de registrar los nuevos nomencladores.
# cargarNomencladoresArchivo(HttpServletRequest request, HttpServletResponse response):void	Método encargado de cargar el archivo y buscar nuevos nomencladores.
GestionarNomencladoresFacade	
Atributos	**Descripción**

- nomencladormanager: GestionarNomencladoresManagerImpl	Manager del módulo Gestionar clasificadores.
Métodos	**Descripción**
+ listarProveedores ():List<Proveedor>	Método encargado de listar todos los Proveedores.
+ listarIntituciones ():List<Institucion>	Método encargado de listar todas las Instituciones
+registrarNomencladoresCargadosProveedores(ProveedorP):void	Método encargado de registrar los nuevos Proveedores.
GestionarNomencladoresManager	
Atributos	
- globalFacade: GlobalFacade	
Métodos	**Descripción**
+ listarProveedores ():List<Proveedor>	Método encargado de listar todos los Proveedores.
+ listarIntituciones ():List<Institucion>	Método encargado de listar todas las Instituciones
+registrarNomencladoresCargadosProveedores(ProveedorP):void	Método encargado de registrar los nuevos Proveedores.
+registrarNomencladoresCargadosInstituciones(InstitucionI): void	Método encargado de registrar las nuevas Instituciones.

Tabla 7: Descripción de las Clases del Diseño

2.2.2.4 Patrones de Diseño empleados

En el capítulo 1 se caracterizan de manera general cada uno de los patrones empleados en el desarrollo de la solución, a continuación se especifica el uso de los patrones seleccionados:

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.

Patrones GRASP

- **Controlador:** Las clases controladoras de los módulos, constituyen un ejemplo de la aplicación de este patrón, las mismas tendrán la responsabilidad de escuchar y responder a las peticiones realizadas por la capa de presentación y de comunicarse con la capa de negocio. Por ejemplo en la interfaz RegistrarMCalendarioSimpleFormController**.**
- **Experto:** Este patrón fue utilizado, en el diseño, en la definición de las clases de acuerdo con las funcionalidades que deben realizar a partir de la información que manejan.
- **Alta cohesión:** Este patrón se evidencia en la agrupación de las clases en dependencia de los requerimientos.
- **Bajo acoplamiento:** Este patrón se manifiesta con la definición de interfaces e implementaciones, como puede ser la interfaz MaquetaCalendarioFacade y su implementación MaquetaCalendarioFacadeImpl permitiendo que clases como RegistrarMCalendarioSimpleFormController se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema, logrando que el código sea más fácil de entender, mantener y reutilizar.

Patrones GoF

- **Fachada:** La utilización de este patrón se evidencia en la definición de la interfaz MaquetaCalendarioFacade responsable de la comunicación entre la capa de presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.
- **Cadena de Responsabilidad:** Cuando desde la vista le solicita información que se encuentra en la base de datos, esta es atendida primeramente por los Controller, luego por la Facade, el Manager y finalmente el DAO, evidenciándose de esta manera la utilización de dicho patrón.

Patrón Modelo Vista Controlador (MVC): Se evidencia su aplicación en la arquitectura dividida en tres capas, utilizada en el proyecto SAGEB. La capa de presentación con la inclusión de la clase RegistrarMCalendarioSimpleFormController, la capa de negocio, con

la inclusión de las clases MaquetaCalendarioFacade y MaquetaCalendarioManager**,** y de acceso a datos, que contiene las interfaces DAO por cada funcionalidad y sus implementaciones.

2.2.3 Validación del Diseño

Un elemento clave de cualquier proceso de ingeniería de software es la medición. Se emplean las medidas para valorar la calidad de los productos de ingeniería o de los sistemas que se construyen. Las métricas técnicas para el software de computadora proporcionan una manera sistemática de valorar la calidad basándose en un conjunto de reglas definidas y al ingeniero del software una visión interna en el acto, lo que le permite descubrir y corregir problemas potenciales, así como, una indicación en tiempo real de la eficacia del análisis, del diseño y de la estructura del código, la efectividad de los casos de prueba, y la calidad global del sistema a construir. (27)

Para la validación del diseño se utilizarán las métricas:

- **Tamaño operacional de clase (TOC)**: Se centran en el recuento de atributos y operaciones para cada clase individual y los valores promedio para el sistema orientado a objetos. 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, por tanto entre menor sea el número de métodos en las clases, mayor es el por ciento de aceptabilidad y calidad en el diseño. (31)
- **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, por tanto mientras menor sea las relaciones entre clases, mayor es el porciento de aceptabilidad y calidad del diseño. (31)

Las métricas TOC y RC posibilitan medir los siguientes atributos de calidad:

- **Responsabilidad:** Responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.
- **Complejidad del mantenimiento:** Nivel de esfuerzo necesario para sustentar, mejorar o corregir el diseño de software propuesto. Puede influir significativamente en los costes y la planificación del proyecto.

- **Complejidad de implementación:** Grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.
- **Acoplamiento:** Dependencia o interconexión de una clase o estructura de clase respecto a otras.
- **Cantidad de pruebas:** Número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto diseñado.

2.2.3.1 Tamaño Operacional de Clase (TOC)

Para la evaluación de las clases fueron utilizados umbrales para el tamaño general, la responsabilidad, la complejidad y la reutilización de las clases.

Atributo de calidad	Aumento del TOC	Categoría	Criterio
Responsabilidad	Implica un aumento de la responsabilidad asignada a la clase	Baja	<= PO*
		Madia	Entre PO y 2*PO
		Alta	> 2*PO
Complejidad de implementación	Implica un aumento de la complejidad de implementación de la clase	Baja	<= PO
		Madia	Entre PO y 2*PO
		Alta	> 2*PO
Reutilización	Implica una disminución del grado de reutilización de la clase	Baja	> 2*PO
		Madia	Entre PO y 2*PO
		Alta	<= PO

Tabla 8: Atributos de calidad que afecta el TOC

Resultados de la métrica TOC:

Durante la evaluación de la métrica TOC los resultados obtenidos demuestran que los atributos de calidad de las clases se encuentran en un nivel satisfactorio; de manera que se puede confirmar la elevada reutilización con un 60% y cómo se reducen la responsabilidad y la complejidad de implementación en un 60%. A continuación el siguiente gráfico recoge los resultados positivos obtenidos y para ver la representación en % de la incidencia de los resultados en cada atributo de calidad ver Anexo3.

* PO: Promedio Operacional

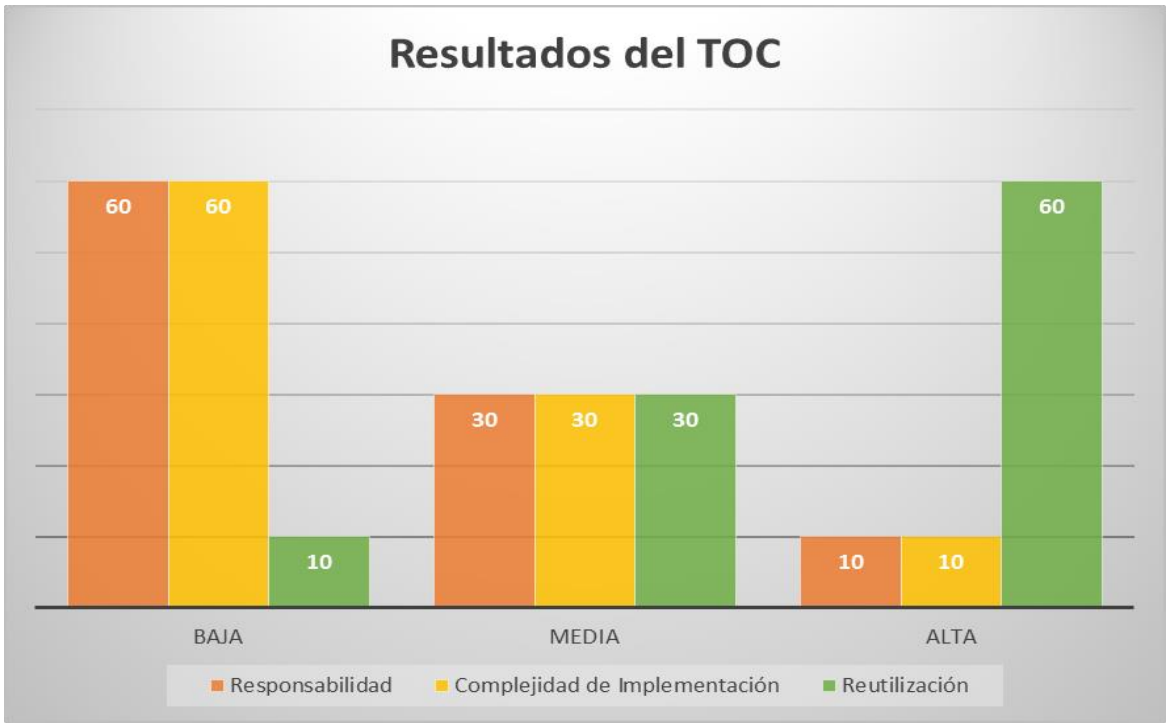


Figura 6: Resultados de la métrica TOC

2.2.3.2 Relaciones entre Clases (RC)

Esta métrica está dada por el número de relaciones de uso de una clase con otra. Para la evaluación de las clases fueron utilizados umbrales para el acoplamiento, complejidad de mantenimiento, la reutilización y cantidad de pruebas.

Atributo de calidad	Aumento del RC	Categoría	Criterio
Acoplamiento	Implica un aumento del acoplamiento de la clase	Ninguno	0
		Bajo	1
		Medio	2
		Alto	>2
Complejidad de mantenimiento	Implica un aumento de la complejidad de mantenimiento de la clase	Baja	<= PO
		Media	Entre PO y 2*PO
		Alta	> 2*PO
Reutilización	Implica una disminución en el grado de reutilización de la clase	Baja	> 2*PO
		Media	Entre PO y 2*PO
		Alta	<= PO
Cantidad de	Implica un aumento de la	Baja	<= PO

pruebas	cantidad de pruebas de unidad necesarias para probar una clase	Media	Entre PO y 2*PO
		Alta	> 2*PO

Tabla 9: Atributos de calidad que afecta las RC

Resultados de la métrica RC:

Luego de realizarse la prueba, los resultados obtenidos demuestran que los atributos de calidad de las clases se encuentran en un nivel satisfactorio; de manera que se puede confirmar la elevada reutilización con un 75%, una baja complejidad de mantenimiento y la cantidad de pruebas en un 75% y un bajo acoplamiento con un 55%. A continuación el siguiente gráfico muestra los resultados obtenidos y para ver la representación en % de la incidencia de los resultados en cada atributo de calidad ver Anexo3a.

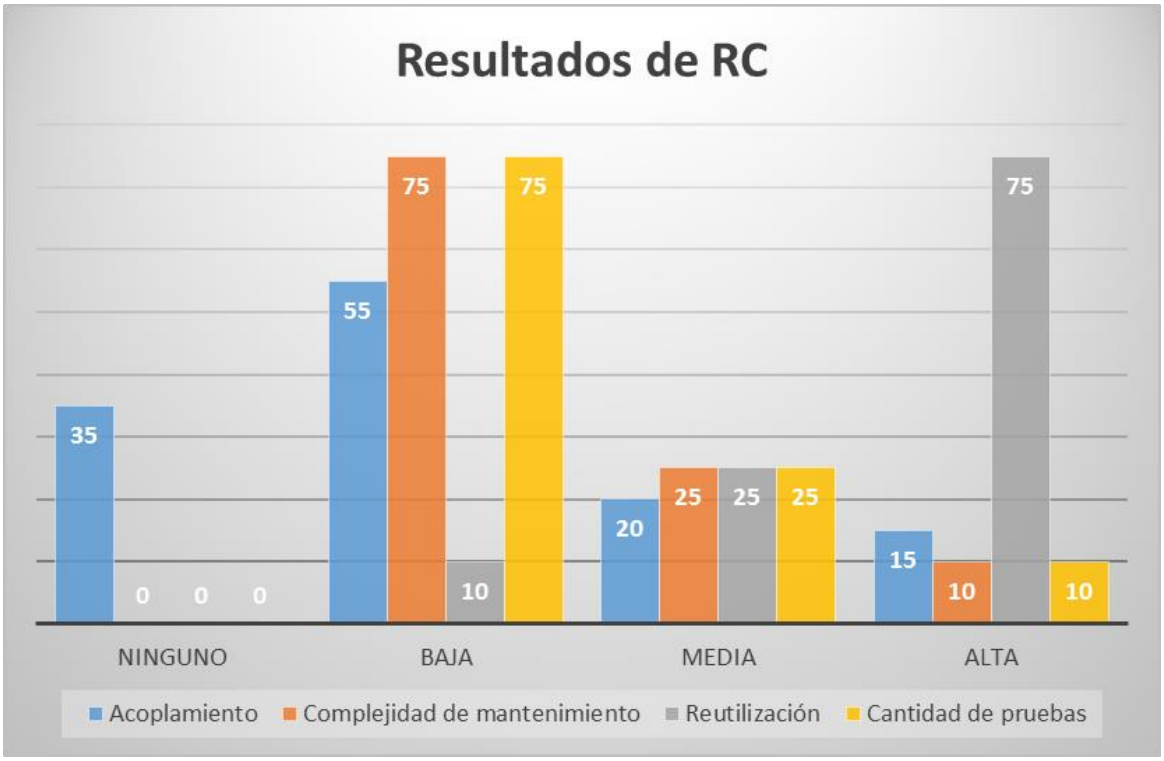


Figura 7: Resultados de la métrica RC

2.3 Conclusiones Parciales

Como resultado de las disciplinas de Requisitos y Análisis y Diseño se obtuvo un modelo de diseño que en conjunto con el uso correcto de los patrones estudiados y la realización de las métricas de evaluación se obtuvo un diseño flexible y escalable fundamental para la disciplina de Implementación.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

El siguiente capítulo abarca las Disciplinas de Implementación, Pruebas Internas y Pruebas de Liberación, dentro de la fase de Desarrollo de los módulos de la solución. Incluye la descripción de las clases y funcionalidades relacionadas, además del modelo de componentes para la separación de los aspectos funcionales y no funcionales del sistema, y la relación de los estándares de codificación utilizados. Se aborda también las pruebas de software a insertar, de caja blanca al código y de caja negra a la interfaz de las funcionalidades a desarrollar.

3.1 Implementación

El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo. La implementación es un proceso de varias fases que empieza cuando se crea una aplicación en el equipo de un desarrollador y termina cuando está instalada y lista para ejecutarse en el equipo de un usuario. En esta etapa se comienza con el resultado de la etapa de Diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, *scripts*, ficheros de código binario, ejecutables y similares.

3.1.1 Estándares de Codificación

Uno de los instrumentos que facilitan la calidad del desarrollo del software es la adopción de estándares de estilo y codificación. Las convenciones de código son de gran importancia ya que permiten asegurar la legibilidad del código entre distintos programadores, facilitando el *debugging*[*] del mismo, proveer una guía para el encargado de mantenimiento y/o actualización del sistema con código claro y bien documentado; y facilitar la portabilidad entre plataformas y aplicaciones. (32)

Para del desarrollo del sistema Quarxo se utilizan las notaciones:

- **PascalCasing:** 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: MaquetaCalendarioManager)
- **CamelCasing:** Para nombrar los métodos y variables que se encuentran en las

[*] Debugging: Depuración.
Depuración de programas es el proceso de identificar y corregir errores de programación.

clases, definiendo que deben comenzar con minúscula y en caso de contener más de una palabra deben comenzar con letra mayúscula. (Ejemplo: listarMCalendario())

3.1.1.1 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 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: **RegistrarMCalendarioSimpleFormController**

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

- ✓ [Nombre de la funcionalidad a la que responde] + [Nombre del paquete]
- ✓ Ejemplo: **MaquetaCalendarioValidator**

- **Capa de Negocio**

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

Para las interfaces:

- ✓ [Nombre del módulo] + [Nombre del paquete]
- ✓ Ejemplo: **MaquetaCalendarioManager**

Para sub-paquetes *impl*:

- ✓ [Nombre del módulo] + [Nombre del paquete] + Impl
- ✓ Ejemplo: **MaquetaCalendarioManagerImpl**.

- **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: **MaquetaCalendarioDAO**

Sub-paquete *impl*:

- ✓ [Nombre de la entidad] + DAOImpl
- ✓ Ejemplo: **MaquetaCalendarioDAOImpl**

3.1.2 Diagrama de Componentes

El Diagrama de Componentes modela los aspectos físicos y la vista de implementación estática de un sistema y muestra un conjunto de componentes y sus relaciones.

A continuación se muestra el diagrama de componentes del subsistema Vencimiento, que contiene el módulo Gestionar Maquetas, donde se puede apreciar la interacción del subsistema y los demás componentes. Los componentes que se relacionan con los demás módulos: Depósitos, Nomencladores y Notificaciones, son los mismos y se comportan exactamente igual como se observa en la figura. (33)

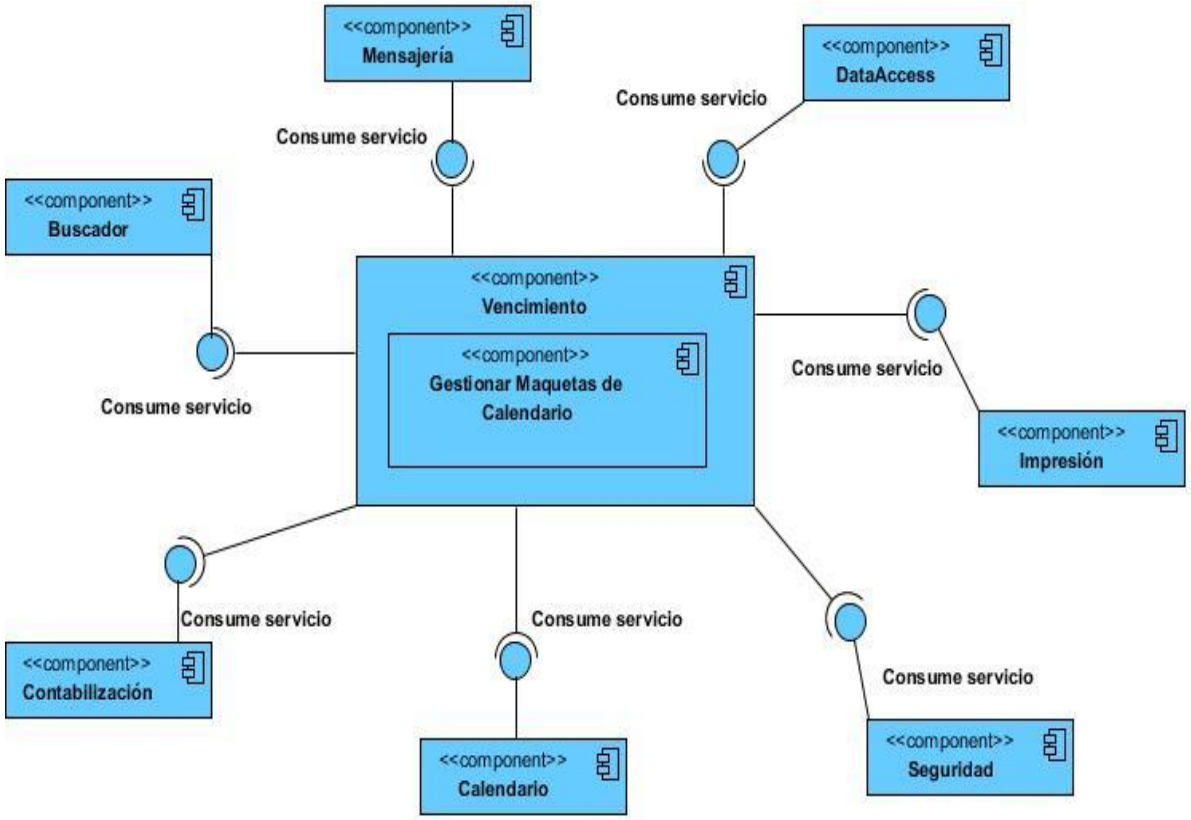


Figura 8: Diagrama de Componentes

Descripción de los Componentes:

Buscador: Mediante este componente se ofrecen un grupo de funcionalidades y clases que forman en su conjunto el motor de búsqueda del sistema, mediante una interfaz gráfica se solicitan los diferentes conceptos y criterios en dependencia del módulo donde se emplee.

Mensajería: Encargado de conceder las clases necesarias para el envío de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos implicados en ella.

DataAccess: Utilizando este componente se acceden a las funciones necesarias para llevar a cabo la conexión a la base de datos.

Impresión: Tiene como función brindar las funcionalidades mediante las cuales es posible imprimir determinada información.

Seguridad: Es el encargado de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos en dependencia del usuario que las realice, gestiona la información de los usuarios, roles y permisos necesarios para lograr la confiabilidad requerida en el sistema.

Calendario: Brinda un conjunto de clases además de una interfaz gráfica, responsables de gestionar la creación de las formas de pago y los vencimientos tanto para los pagos principales como para los intereses, relacionada con una determinada operación bancaria.

Contabilización: Propone un grupo de clases necesarias para realizar la contabilización de determinadas operaciones que así lo requieran y otro grupo de clases y una interfaz gráfica para el cobro de comisiones que se asocien a determinada operación.

3.1.3 Diagrama de Despliegue

El Diagrama de Despliegue modela los aspectos físicos y la vista de despliegue estática de un sistema, así como la topología del hardware donde se ejecuta el sistema. (33)

A continuación se muestra el diagrama de despliegue del sistema Quarxo.

Figura 9: Diagrama de Despliegue

3.2 Pruebas

La etapa de pruebas es una de las más costosas del ciclo de vida del software. En sentido estricto, deben realizarse pruebas a todos los artefactos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, diagramas de diversos tipos, el código fuente y el resto de productos que forman parte de la aplicación. (34)

3.2.1 Pruebas Internas

Durante esta fase se desarrollan las pruebas del grupo de calidad del centro verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software, es decir, requisitos que el producto debería cumplir y que aún no cumple. (7)

3.2.1.1 Pruebas de Caja Blanca

Las pruebas estructurales o de caja blanca del software se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. (27)

Denominada a veces prueba de caja de cristal, la prueba de caja blanca es un método mediante el cual el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo, que se practiquen todas las decisiones lógicas en sus vertientes verdadera y falsa, que se ejecuten todos los bucles en sus límites y con sus límites operacionales y que se ejerciten

las estructuras internas de datos para asegurar su validez. (27)

Aplicación de la Prueba de Caja Blanca:

Para la realización de esta prueba se hace uso del framework **JUnit** el cual se puede integrar al IDE de desarrollo Eclipse. Este framework permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

A continuación, para la aplicación de la prueba, se toma como caso de estudio el método **eliminarMCalendario(HttpServletRequest request, HttpServletResponse response)** perteneciente a la clase **CargarDatosMCalendarioMultiActionController**.

```
public void eliminarMCalendario(HttpServletRequest request,
        HttpServletResponse response) {
    String idMaqueta = request.getParameter("ref");
    String detalles = "";
    String idUser = "";

    MaquetaCalendario maqueta = null;
    try {
        int id = Integer.valueOf(idMaqueta);
        maqueta = maquetaFacade.obtenerMCalendarioById(id);
        idUser = maqueta.getIdUsuario();
    } catch (Exception e) {
    }

    if (maqueta != null
            && idUser.equals(UserHandler.getUser().getCodUsuari())) {
        maquetaFacade.eliminarMCalendario(maqueta);
        detalles = "m0";
    } else {
        detalles = "m1";
    }

    JSONObject json = new JSONObject();
    json.put("detalles", detalles);

    try {
        ResponseUtil.escribirDatosEnElResponse(response, json.toString());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 10: Método a probar por el framework JUnit

Para probar dicho método primeramente se crea una clase, nombrada en este caso como **TestJUnit,** la cual debe extender de **TestCase**, como se observa y describe a

continuación.

```
public class TestJUnit extends TestCase {
private MockControl controlhttpServlet;
private HttpServletRequest mockHttpServletRequest;

private  MockControl controlHttpResponse;
private HttpServletResponse mockHttpServletResponse;

private MockControl controlHttpSession;
private HttpSession mockhttpSession;

CargarDatosMCalendarioMultiActionController cargarDatosMaqueta;


@SuppressWarnings("deprecation")
protected void setUp() throws Exception{
   cargarDatosMaqueta = new CargarDatosMCalendarioMultiActionController();

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

MaquetaCalendarioManagerImpl manager = (MaquetaCalendarioManagerImpl) context.getBean("maquetaManager");
MaquetaCalendarioFacadeImpl facade = new MaquetaCalendarioFacadeImpl();
facade.setMaquetaManager(manager);
cargarDatosMaqueta.setMaquetaFacade(facade);

   controlhttpServlet = MockControl.createControl(HttpServletRequest.class);
   mockHttpServletRequest = (HttpServletRequest)controlhttpServlet.getMock();
   controlHttpResponse = MockControl.createControl(HttpServletResponse.class);
   mockHttpServletResponse = (HttpServletResponse)controlHttpResponse.getMock();
   controlHttpSession = MockControl.createControl(HttpSession.class);
   mockhttpSession = (HttpSession)controlHttpSession.getMock();
   super.setUp();

}
```

Figura 11: Clase TestJUnit

Después de creada la clase, se crea un objeto de la clase en la cual se encuentra el método que se va a probar y dos simulacros de los parámetros que se le pasa a dicho método, uno de **HttpServletRequest** y otro de **HttpServletResponse**, con sus respectivos **Mock** y **Control** 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 de la operación que se va a probar.

```
protected void tearDown() throws Exception {
    super.tearDown();
}

public void testEliminarMaquetaCalendario() {
      mockHttpServletRequest.getParameter("ref");
      controlhttpServlet.setReturnValue("100");
      controlhttpServlet.replay();
 boolean detalle=cargarDatosMaqueta.eliminarMCalendario(mockHttpServletRequest, mockHttpServletResponse);
 assertTrue("Comprobar", detalle);
}
```

Figura 12: Métodos tearDown y testEliminarMaquetaCalendario de la clase TestJUnit

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

Seguidamente se crea un método, **testEliminarMaquetaCalendario()**, en donde se especifica cómo son recibidos los parámetros y cuál es el identificador de cada uno, en el método que será sometido a prueba: **eliminarMCalendario(HttpServletRequest request, HttpServletResponse response)**.

Posteriormente mediante el método **assertTrue()** se ejecuta el procedimiento que está siendo analizado y se espera que el framework termine el análisis. Según el resultado de la prueba, JUnit muestra una ventana: si es satisfactorio se evidencia a través de una línea de color verde, en caso contrario de color rojo. A continuación se muestra el resultado de dicha prueba.

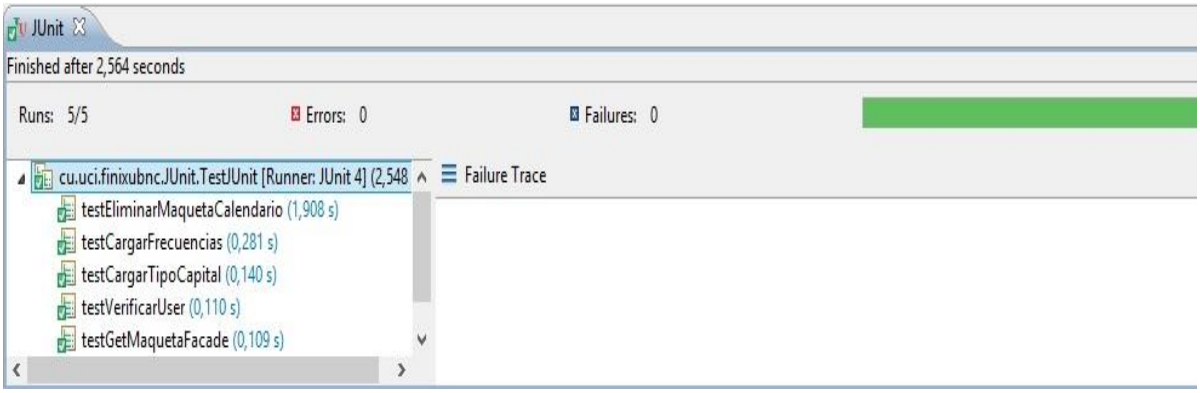


Figura 13: Resultados de la prueba con JUnit

Ver demás resultados las pruebas en Anexo4.

3.2.2 Pruebas de Liberación

Se aplican pruebas diseñadas e implementadas por el Laboratorio Industrial de Pruebas de Software a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación. (7)

3.2.2.1 Pruebas de Caja Negra

Las pruebas funcionales o de caja negra se centran en los requisitos funcionales del software. Permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra intenta encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a base de datos externas, errores de rendimiento y errores de inicialización y terminación. (27)

Mediante las técnicas de prueba de caja negra se obtiene un conjunto de casos de prueba que satisfacen los siguientes criterios: casos de prueba que reducen, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable y casos de prueba que dicen algo de la presencia o ausencia de clases de errores en lugar de errores asociados solamente con la prueba que se está realizando. (27)

Aplicación de la Prueba de Caja Negra:

Particiones de Equivalencia es la técnica empleada para confeccionar los casos de prueba de Caja Negra, la misma es 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. A continuación se muestra la aplicación de la técnica en el requisito funcional **Actualizar Maqueta de Calendario**.

Caso de Prueba del Requisito Funcional Actualizar Maqueta de Calendario	
Descripción General:	
El sistema debe permitir actualizar las maquetas de calendario que el usuario desee, para ello se deben actualizar los campos correspondientes a su definición, de cancelarse esta operación, no se efectuará la actualización de la maqueta de calendario.	
Condiciones de Ejecución:	
1. Se debe identificar y autenticar ante el sistema y tener los permisos para ejecutar esta acción. 2. Se debe seleccionar el subsistema Vencimiento y dentro de este el módulo Gestionar Maquetas. 3. Se debe seleccionar al desplegarse el menú, la opción: Buscar.	
Escenarios de pruebas	**Flujo del escenario**
EP 1.1: Actualizar una maqueta de calendario correctamente.	1. El usuario selecciona la maqueta que desea actualizar. 2. Se presiona el botón Actualizar**.** 3. Se introducen los nuevos datos de la maqueta. 4. Se presiona el botón Aceptar. 5. El sistema muestra el mensaje Operación realizada satisfactoriamente. 6. Se presiona el botón Aceptar del mensaje.
EP 1.2: Actualizar una maqueta de calendario con datos inválidos.	1. El usuario selecciona la maqueta a actualizar. 2. Se presiona el botón Actualizar. 3. Se introducen datos inválidos de la maqueta. 4. Se presiona el botón Aceptar. 5. El sistema señala aquellos campos de llenado obligatorio a través del mensaje de alerta: El valor especificado no es válido. 6. El usuario corrige los datos. 7. Se presiona el botón Aceptar**.** 8. El sistema muestra el mensaje: Operación realizada satisfactoriamente. 9. Se presiona el botón Aceptar del mensaje.

EP 1.3: Actualizar una maqueta de calendario dejando campos requeridos en blanco.	1. El usuario no selecciona ninguna maqueta para actualizar. 2. Se presiona el botón Actualizar. 3. El sistema muestra el mensaje: Debe seleccionar la maqueta que desea actualizar. 4. El usuario presiona el botón Aceptar del mensaje y selecciona la maqueta para actualizar. 5. Se presiona el botón Actualizar. 6. Se introducen los nuevos datos de la maqueta dejando campos obligatorios en blanco. 7. Se presiona el botón Aceptar. 8. El sistema señala aquellos campos de llenado obligatorio a través del mensaje de alerta: El valor especificado no es válido. 9. El usuario corrige los datos. 10. Se presiona el botón Aceptar. 11. El sistema muestra el mensaje: Operación realizada satisfactoriamente. Se presiona el botón Aceptar del mensaje.
EP 1.4: Cancelar la actualización de un maqueta de calendario.	1. Se muestra el mensaje: ¿Está seguro que desea cancelar la operación? 2. Si se acepta el mensaje se concluye el requisito, sino no se realiza ninguna operación.

Tabla 10: Escenarios de Prueba del RF Actualizar Maqueta de Calendario

Nro.	Nombre de campo	Tipo	Válido	Inválido
1	Nombre	Campo de texto	Se introduce el nombre de la maqueta de calendario	No puede ser nulo.
2	Descripción	Campo de texto	Se introduce una descripción de la maqueta de calendario	No admite caracteres especiales.
3	Formas de pago	Lista desplegable	Valores predeterminados. Se cargan las formas de pago existentes en el sistema, para la selección	Todos los valores que no estén en la lista

			de una de estos.	predeterminada.
4	Cantidad	Campo de texto	Se introducen la cantidad de formas de pago de un tipo específico.	No puede ser nulo. No admite caracteres especiales y/o letras.
5	Tasa estimada	Campo de texto	Se introduce la tasa estimada para el cálculo de los intereses	No puede ser nulo. No admite caracteres especiales y/o letras.
6	Tasa Spread	Campo de texto	Se introduce la tasa spread para el cálculo de los intereses.	No puede ser nulo. No admite caracteres especiales y/o letras.
7	Capital	Lista desplegable	Se cargan los tipos de Capital existentes en el sistema, para la selección de uno de estos.	Todos los valores que no estén en la lista predeterminada.
8	Cantidad de días	Lista desplegable	Se cargan los valores de Cantidad de días existentes en el sistema, para la selección de una de estos.	Todos los valores que no estén en la lista predeterminada.
9	Base	Campo de texto	Se introduce la base para el cálculo de los intereses.	No puede ser nulo. No admite caracteres especiales y/o letras.
10	Interés	Campo de texto	Se introduce la porciento de interés para el cálculo de los intereses.	No puede ser nulo. No admite caracteres especiales y/o letras.

Tabla 11: Descripción de las Variables de Entrada

Id del escenario	EP 1.1	EP 1.2	EP 1.3	EP 1.4
Escenario	Crear una maqueta de calendario correctamente	Crear una maqueta de calendario con datos inválidos	Crear una maqueta de calendario dejando campos requeridos en blanco.	Cancelar la actualización de un maqueta de calendario
Nombre	V(MC Venezuela)	I()	V(MC Venezuela)	V(MC Venezuela)
Formas de pago	V(Semestral)	I(Cuadro)	V(Semestral)	V(Semestral)
Cantidad	V(10)	I(Diez)	V(10)	V(10)
Tasa estimada	V(10)	I(Diez)	V(10)	V(10)
Tasa Spread	V(10)	I(Diez)	V(10)	V(10)
Capital	V(Sobre saldo)	I(Todo)	V(Sobre saldo)	V(Sobre saldo)
Cantidad de días	V(Último vencimiento)	I(Vencimiento)	V(Último vencimiento)	V(Último vencimiento)
Base	V(360)	I(Tres)	V(360)	V(360)
Interés	V(1.5)	I(Uno)	V(1.5)	V(1.5)
Respuesta del sistema	El sistema registra los datos introducidos de la nueva maqueta y muestra el mensaje: Operación realizada satisfactoriamente.	El sistema señala aquellos campos de llenado obligatorio a través del mensaje de alerta: El valor especificado no es válido.	El sistema señala aquellos campos de llenado obligatorio a través del mensaje de alerta: El valor especificado no es válido.	El sistema muestra el mensaje de alerta: ¿Está seguro que desea cancelar la operación?
Resultado de la prueba	Satisfactoria	Satisfactoria	Satisfactoria	Satisfactoria

Tabla 12: Juegos de Datos a Probar

Como resultado se obtuvieron valores satisfactorios en el conjunto de casos de pruebas desarrollados. Para consultar demás casos de pruebas realizados ver Anexo5.

3.3 Validación de las Variables de la Investigación

El perfeccionamiento de las operaciones con el desarrollo de la solución, lo determina la agilidad y facilidad con que puedan ser realizados los procesos de creación de calendario, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo, como se planteaba en la idea a defender al inicio de la investigación, lo que ocasiona una reducción de tiempo y complejidad en la ejecución. A continuación la solución se somete a pruebas para comprobar que en efecto si se agilizan y facilitan las tareas antes mencionadas.

Para la demostración se toman dos casos de estudio: Registrar Calendario y Enviar Notificación.

La operación de Registrar Calendario involucra la configuración de las formas de pagos, el período de gracia y los intereses que se generan, en caso de este último que implica el registro de datos para cálculos financieros complejos. Los operadores del BNC actualmente deben registrar cada calendario de inicio a fin cada vez que acceden a este componente, ralentizando esta tarea y haciéndose engorrosa para el usuario. Mediante las nuevas funcionalidades implementadas, el usuario tendrá la facilidad de guardar las configuraciones de un calendario creado anteriormente y cargar esos datos registrados las veces que le sean necesarias para la confección de uno o más calendarios, reduciendo así la cantidad de campos obligatorios a llenar.

En caso de la operación Enviar Notificación hasta el presente el administrador del sistema debe pasar por cada puesto de trabajo para alertar o informar acerca de la ocurrencia de ciertos eventos que están por pasar, como puede ser el aviso del cierre de la aplicación por algún motivo especial, o que dentro de determinado tiempo se realizará el cierre contable. Con el desarrollo de las nuevas funcionalidades, el operador tendrá la posibilidad de enviar notificaciones a un usuario determinado o a todos los usuarios, y dicha notificación puede implicar el cerrado de la sesión correspondiente o incluso del navegador web a los destinatarios de la misma. Por la frecuencia de uso de algunos mensajes, el sistema también brindará la facilidad de poder gestionar su ciclo de vida, de manera que sirvan como plantillas para el envío de los mencionados avisos.

De manera general, a continuación en la siguiente tabla se pone en evidencia la facilidad que proporcionan ambas funcionalidades implementadas a los procesos anteriormente descritos.

Operación	Cantidad de actividades a realizar	
	Antes	**Después**
Registrar Calendario	1. Acceder al componente Calendario. 2. Registrar un calendario de pago llenando los campos requeridos: 2.1 Formas de pago - Período de gracia: 2.2 Días 2.3 Meses 2.4 Años - Intereses: 2.5 Tasa estimada 2.6 Tasa Spread 2.7 Capital 2.8 Cantidad de Días 2.9 Forma de Pago 2.10 Cantidad de formas de pago 2.11 Base 2.12 Interés	1. Acceder al componente Calendario y seleccionar la opción Cargar Maqueta. 2. Seleccionar la maqueta a cargar. 3. Cambiar los datos que el usuario desee.
Enviar Notificación	1. Pasar por cada departamento u oficina del BNC para notificar sobre el evento que está por pasar: 1.1 Depósitos 1.2 Deuda 1.3 Negociaciones 1.4 Préstamos 1.5 Tesorería 1.6 Contabilidad	1. Acceder al módulo Notificaciones y seleccionar la opción de Enviar Notificación. 2. Registrar una notificación llenando los campos requeridos: 2.1 Usuario destino 2.2 Notificación predefinida

	1.7 Administración 1.8 Carta de crédito 1.9 Vencimiento 1.10 Conciliaciones 1.11 Mensajería 1.12 Transferencia 1.13 Títulos valores	2.3 Frecuencia 2.4 Número de iteraciones 2.5 Acción a ejecutar 2.6 Texto de la notificación

Tabla 13: Tabla demostrativa para variable Facilidad

Para demostrar cuanto se han agilizado las operaciones correspondientes a los casos de estudios explicados se presenta a continuación en la siguiente tabla comparativa el tiempo de ejecución de ambas operaciones.

Operación	Cantidad a realizar	Antes	Después
Registrar Calendario	5	50-60 min	10-15 min
Enviar Notificación	5	50-60 min	3-5 min

Tabla 14: Tabla demostrativa para variable Agilidad

Los datos recogidos en la tabla muestran que efectivamente se ha disminuido considerablemente el tiempo de ejecución de ambas operaciones con el desarrollo de las nuevas funcionalidades.

3.4 Conclusiones Parciales

Con la finalización de esta etapa se concluye que las nuevas funcionalidades implementadas y probadas permiten llevar a cabo eficientemente los procesos involucrados con las operaciones de maqueta de calendario, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el BNC.

CONCLUSIONES GENERALES

Con el desarrollo del presente trabajo de diploma se llevaron a cabo todas las tareas a fin de cumplir con los objetivos trazados concluyendo:

- La realización de la fundamentación teórica a través del análisis de los sistemas informáticos de gestión bancaria evidenció la necesidad de desarrollar las funcionalidades implementadas para la segunda fase del sistema Quarxo.
- Las disciplinas de Requisitos, Análisis y Diseño proporcionaron modelos, los cuales fueron validados a través de técnicas y métricas para probar su calidad y eficiencia, necesarios para la implementación de las funcionalidades.
- La realización de las disciplinas Implementación, Pruebas Internas y Pruebas de Liberación brindaron como resultados funcionalidades que permitirán mejorar eficientemente los procesos de creación de maquetas de calendario, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo.
- La validación con el cliente del desarrollo de los módulos Calendario, Notificaciones y fase 2 de los módulos Depósitos y Nomencladores del sistema Quarxo demostró cómo han agilizado y facilitado las operaciones en el BNC.

De manera general se dio solución al problema a resolver mediante el análisis, diseño, implementación y validación de las nuevas funcionalidades, las cuales permitirán agilizar y facilitar los procesos de creación de maquetas de calendario, renovación de depósitos, gestión de nomencladores y envío de notificaciones en el sistema Quarxo.

RECOMENDACIONES

- Aprovechar el estudio realizado en esta investigación para futuras versiones que se realicen del sistema Quarxo y para el desarrollo de otros sistemas de gestión bancaria.

BIBLIOGRAFÍA

1. **Madrazo, Annier Puig.** *Desarrollo del subsistema Vencimiento de Quarxo v2.0.* 2012.
2. **Yoan Antonio Lopez Rodríguez, Yulier Matias León.** *Definición de los requerimientos funcionales del módulo tesorería, préstamos y depósitos del proyecto Banco Nacional.* 2008.
3. **Lázaro J. Blanco Encinosa.** *Apuntes para una historia de la Informática en Cuba. Consideraciones técnicas y organizativas y económicas.*
4. **Piñeiro, Rolando Choy.** *Diseño e Implantación del Subsistema Carta de Créditos Segunda Fase del Sistema QUARXO.* 2012.
5. **Byte.** Byte. [En línea] http://www.bytesw.com/new/sistema_bancario_financierobyte.asp.
6. **Nasoft.** Nasoft . [En línea] http://www.nasoft.com/site/Home/Soluciones/Porproducto/Temenos/tabid/147/Default.aspx
7. **Obregón, Ing. William González.** *Modelo de Desarrollo de Software.* 2012.
8. **Prieto, Félix.** *Patrones de diseño.* Departamento de Informática, Universidad de Valladolid : s.n., 2008 .
9. **Departamento de Lenguajes y Sistemas Informáticos.** *Patrones de Asignación de Responsabilidades (GRASP).* Universidad de Sevilla : s.n.
10. **Santana, Manuel Alejandro Borroto.** *Diseño e Implementación de los subsistemas Créditos y Depósitos del sistema Quarxo para el Banco Nacional de Cuba.* 2011.
11. **Daylen de la Caridad Barban Reyes, Roberto Rodríguez Rodríguez.** *Diseño e Implementación del subsistema Vencimiento del sistema Quarxo para el Banco Nacional de Cuba.* 2011.
12. **Ahsanul Bari, Anupom Syam.** *CakePHP Application Development. Step-by-step introduction to rapid web development using the open-source MVC CakePHP framework.* 2008.
13. **Bizagi Process Modeler.** *Bizagi BPMN 2.0 Ejemplo.*
14. **Erika Camacho, Fabio Cardeso Gabriel Nuñez.** *Arquitectura de Software. Guía de estudio.* 2004.
15. **Zukowski, John.** *Programación Java 2 J2SE 1.4 .* s.l. : Anaya Multimedia, 2003.
16. **Cay S. Horstman, Gary Cornell.** *Core Java 2 Volume I-Fundamentals.* 2000.
17. **Camy, Lázaro Issi.** *JavaScript.* s.l. : Anaya Multimedia, 2002 .

18. **Gavin King, Chritian Bauer, Max Rydahl Anderesen, Emmanuel Bernard, Steve Ebersole.** Hibernate Community Documentation. *HQL: el lenguaje de consulta de Hibernate.* [En línea] 15 de 9 de 2010. http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html/queryhql.htm.

19. **Subrahmanyam Allamaraju, Cedric Beust, John Davies, Tyler Jewell, Rod Johnson, Andy Longshaw, Ramesh Nagappan, Dr.P.G.Sarang, Alex Toussaint, Gary Watson, Alan Williamson.** *Programación Java Server con J2EE Edición 1.3.* s.l. : Anaya Multimedia.

20. **Ramalho, José Antonio.** *SQL Server 7 Iniciación y Referencia.* s.l. : Makron Books, 2000 .

21. **Adriana Rendón Artola, Manuel Alejandro Castellanos Pérez.** *Modelación y construcción de los componente s de actividades y Calendario para el Subsistema Planificación de Objetivos del Sistema Integral de Gestión de Entidades Cedrux.* 2011.

22. **Dave Minter, Jeff Linwood.** *Pro Hibernate 3.* s.l. : Apress, 2005 .

23. **Harmon, James E.** *Using the Dojo JavaScript Library to Build Ajax Applications.* 2009.

24. **Craing Walls, Ryan Breidenbach.** *Spring in Action.* 2005.

25. **Thomas Van de Velde, Bruce Snyder, Christian Dupuis, Sing Li, Anne Horton y Naveen Balani.** *Beginning Sping Framework 2.*

26. **Gary Mak, Josh Long, Daniel Rubio.** *Spring Recipes. A problem-solution approach.* 2010.

27. **Pressman, Roger S.** *Ingeniería del software. Un enfoque práctico. Cuarta edición.* 1998.

28. **Navarro, Javier Mendoza.** *Diseño del Sistema de Tarjeta de Crédito con UML.* s.l. : Universidad Nacional Mayor de San Marcos, Perú .

29. **Gutiérrez, Damián.** *UML Diagramas de Paquetes.* Universidad de los Andes : s.n., 2009.

30. **Vidal, Mari Carmen Otero.** *Diagramas de Clases.* Dpto. de Lenguajes y Sistemas Informáticos, ITESCAM : s.n.

31. **Evelio Clavel Rosales, Dariel Membribes Rodriguez.** *Subsistema Mensajería SLBTR para el sistema Quarxo.* 2012.

32. **Dirección General de Gobierno Digital.** *Estándares de codificaciones de sistemas.* Ministerio de Coordinación de Gabinete, Gobierno del Chubut : s.n., 2006.

33. **Daniele, Ing. Marcela.** *Teoría 11: El Arte de modelar, UML, Diagrama de*

Componentes y Diagrama de Despliegue. UNRC : s.n., 2007.

34. **Usaola, Dr. Macario Polo.** *Mantenimiento Avanzado de Sistemas de Información. Pruebas del software.* Departamento de Informática, Paseo de la Universidad, Ciudad Real : s.n.

GLOSARIO DE TÉRMINOS

- **Campo:** Se refiere campo como al atributo que forma parte del valor de un elemento dentro de un mensaje XML, describe generalmente una propiedad o atributo de un objeto del dominio del negocio.
- **Código abierto (*Open Source*):** Término con el que se conoce al software distribuido y desarrollado libremente.
- **Command:** Es la representación específica de la información con la cual el sistema opera. Se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. Los *command* también pueden operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Framework**: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
- **HTML:** Lenguaje de marcado predominante 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.
- **JSP (*Java Server Page*):** Es una tecnología Java para crear contenido dinámico para web en forma de documentos HTML o XML.
- **Módulo:** cada módulo es una parte del sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.
- **Plugins:** Un *plugin* es un módulo de *hardware* o *software* que adiciona una característica o un servicio específico a un sistema existente.
- **Servlet**: Son objetos que están presentes dentro del contexto de un contenedor de servlets como el Tomcat, es comúnmente utilizado para generar páginas web de forma dinámica a partir de parámetros de una petición de un navegador web.
- **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.

ANEXOS

Anexo 1: Prototipos no funcionales

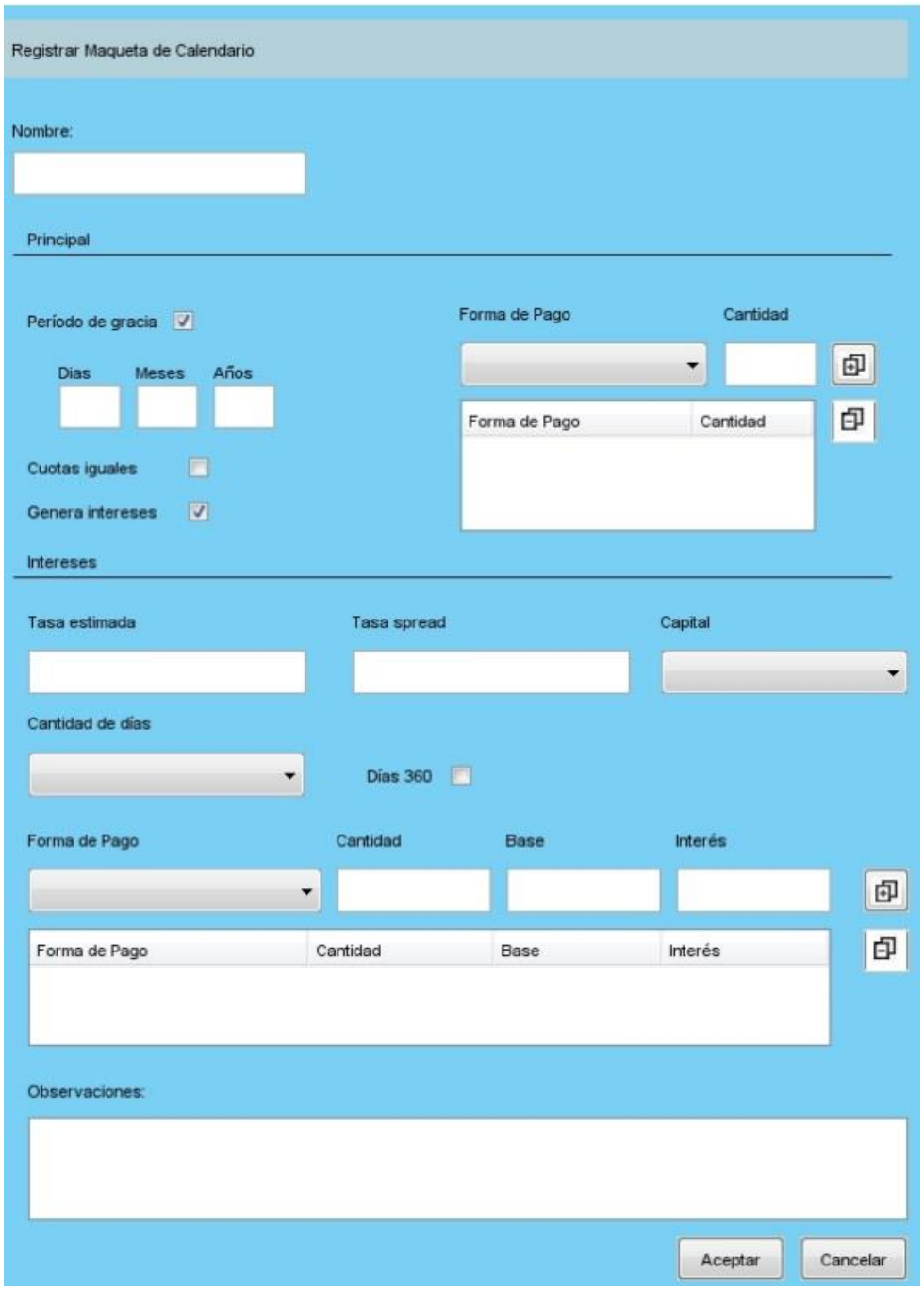


Figura 14: Prototipo no funcional Registrar Maqueta de Calendario

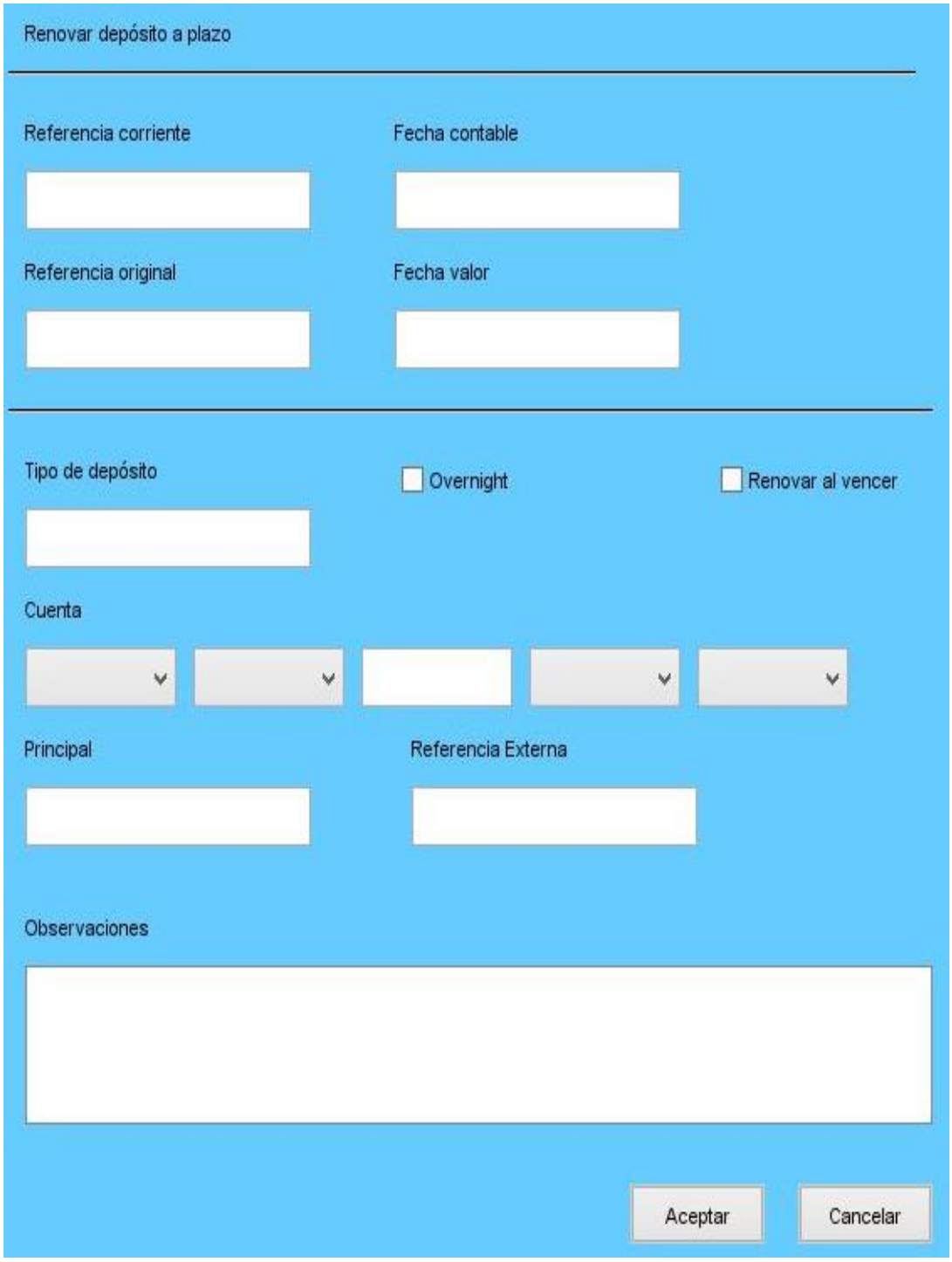


Figura 15: Prototipo no funcional Renovar Depósito a plazo

Anexo 2: Modelo de Diseño

Modelos de Datos:

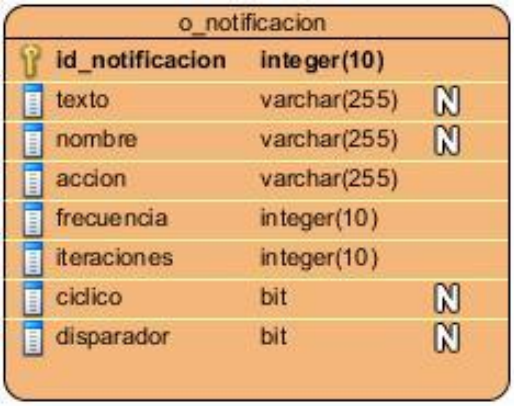


Figura 16: Modelo de Datos del Módulo Notificaciones

Diagrama de Paquetes:

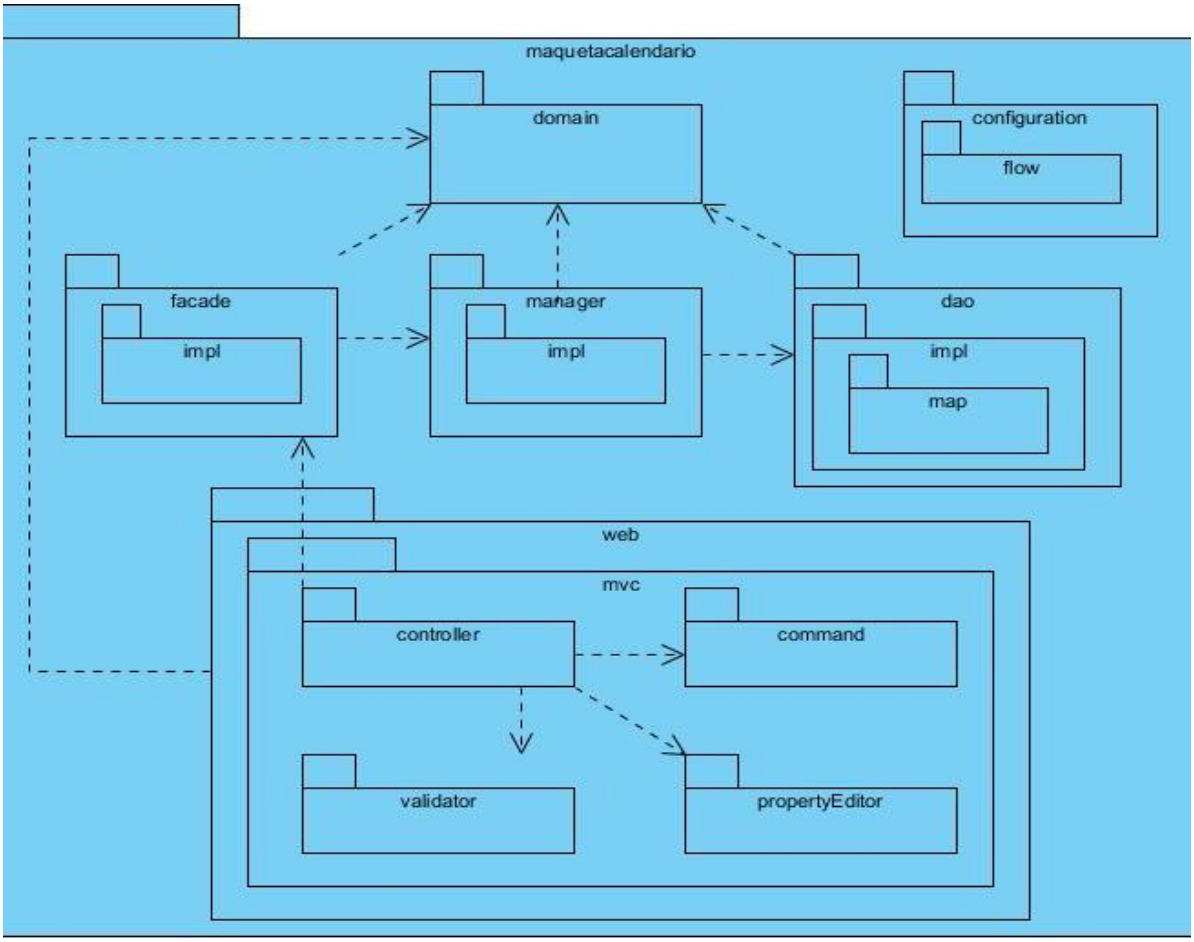


Figura 17: Diagrama de Paquetes del Módulo Gestionar Maquetas

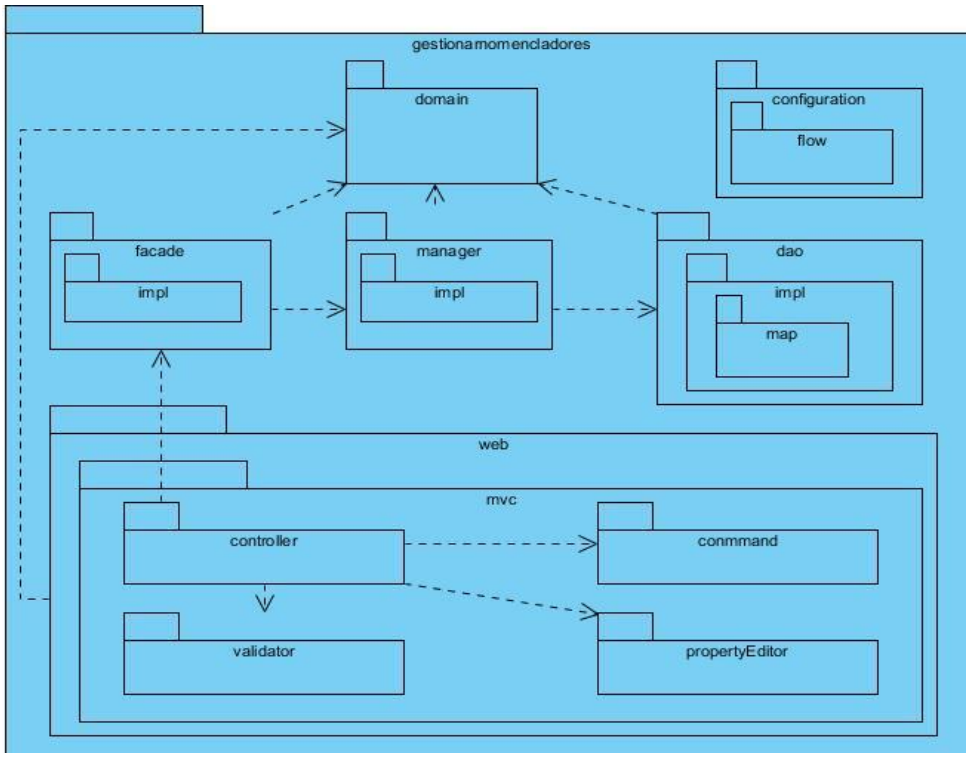


Figura 18: Diagrama de Paquetes del Módulo Gestionar Nomencladores

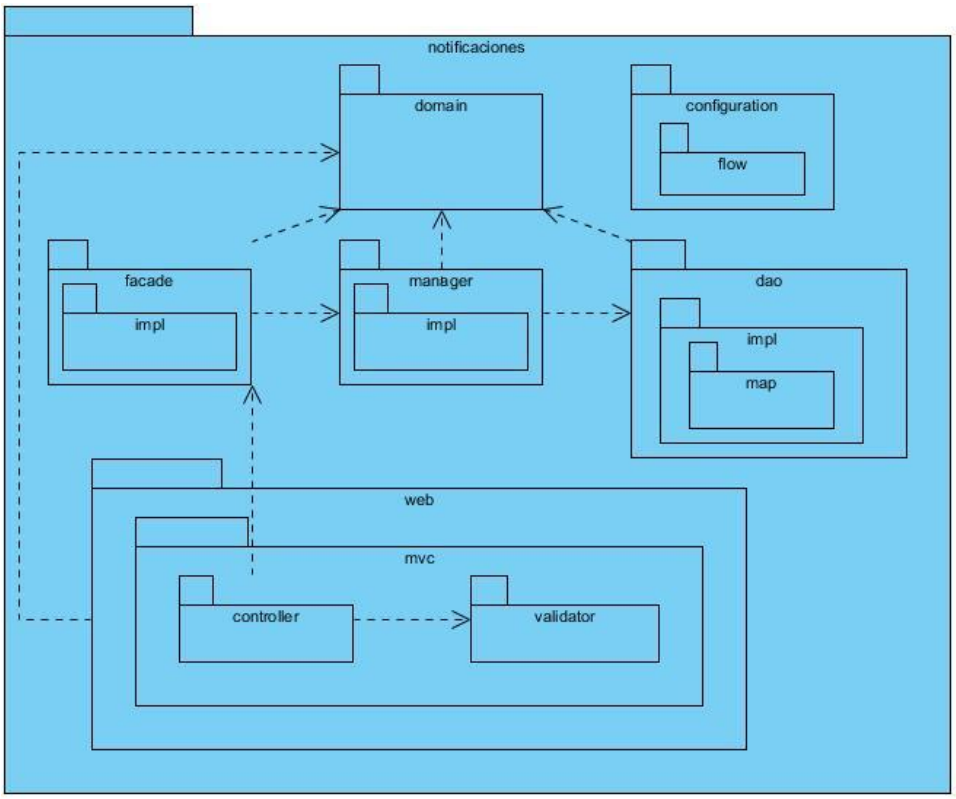


Figura 19: Diagrama de Paquetes del Módulo Notificaciones

Diagrama de Clases del Diseño:

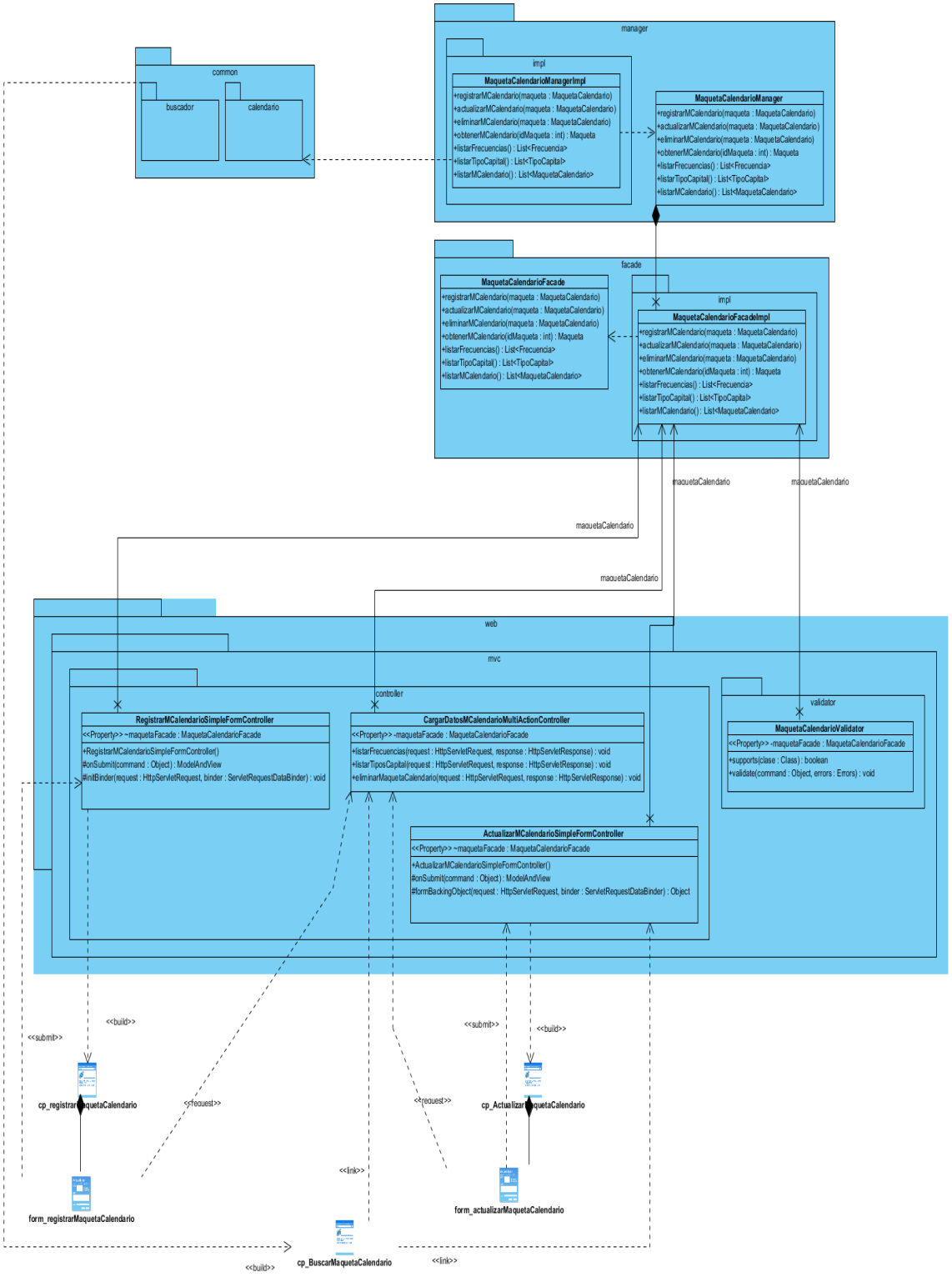


Figura 20: Diagrama de Clases del Diseño del Módulo Gestionar Maquetas

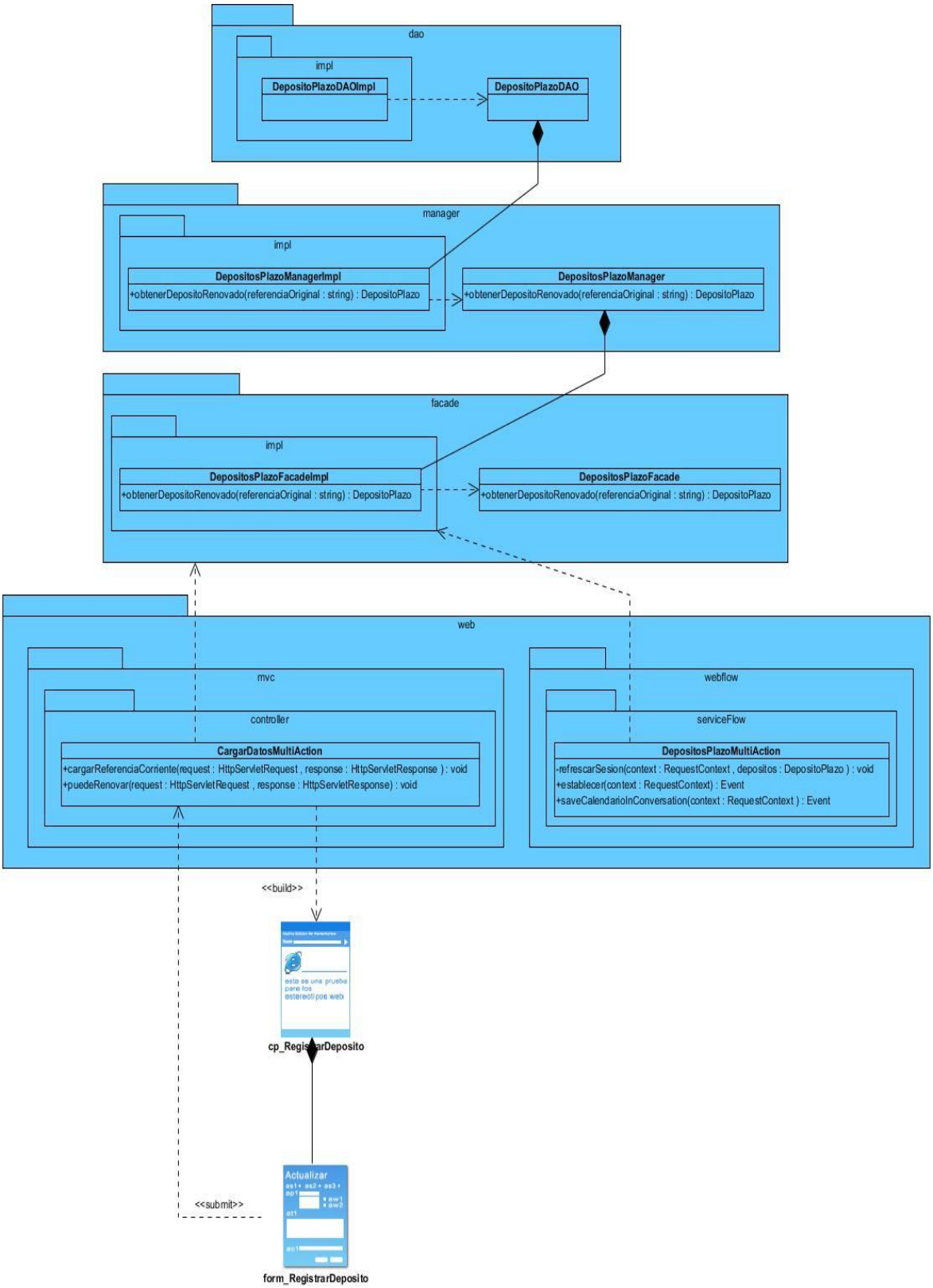


Figura 21: Diagrama de Clases del Diseño del Subsistema Depósitos

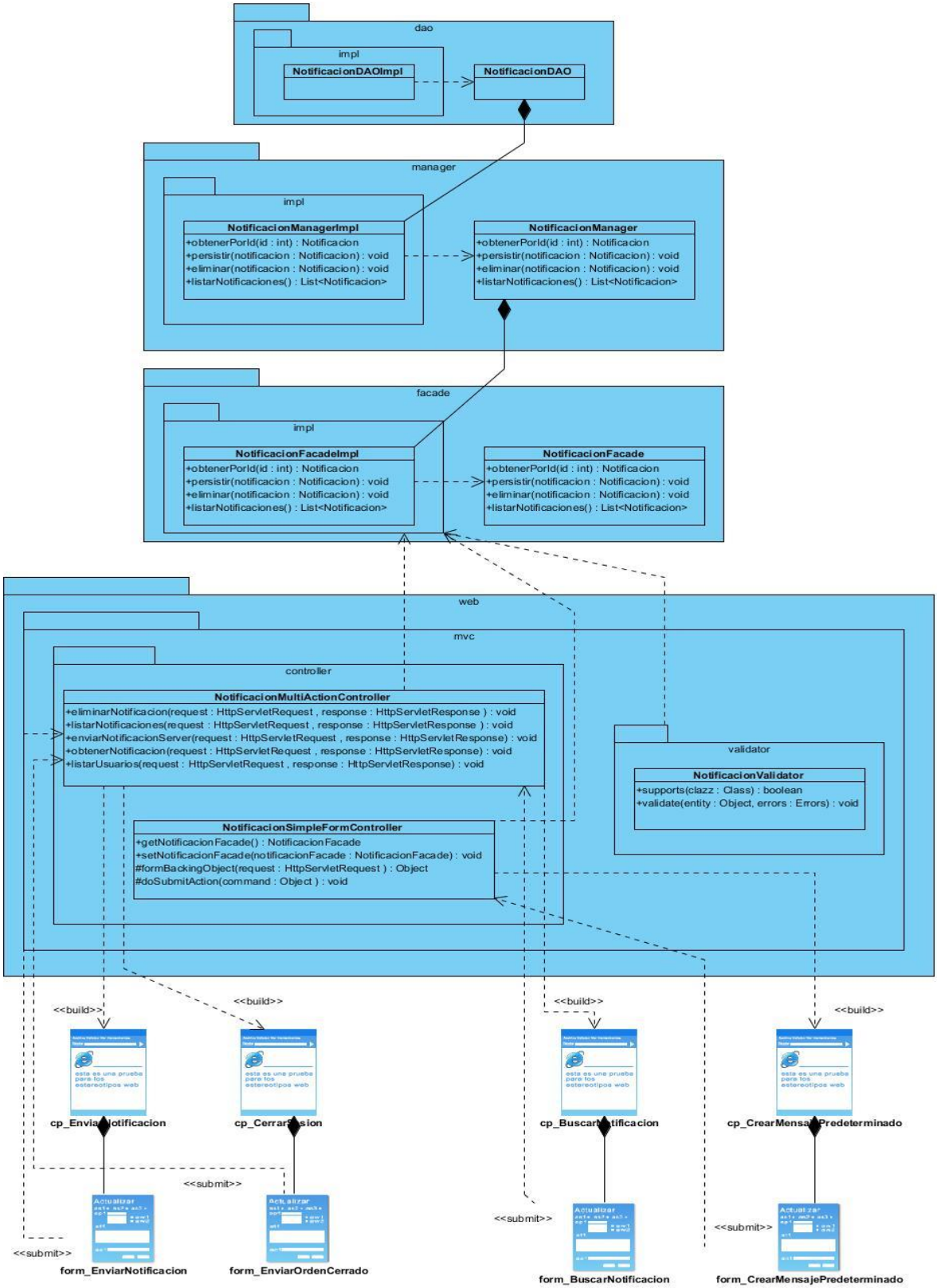
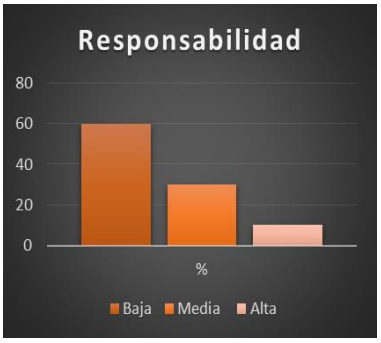


Figura 22: Diagrama de Clases del Diseño del Módulo Notificaciones

Anexo 3: Validación del diseño

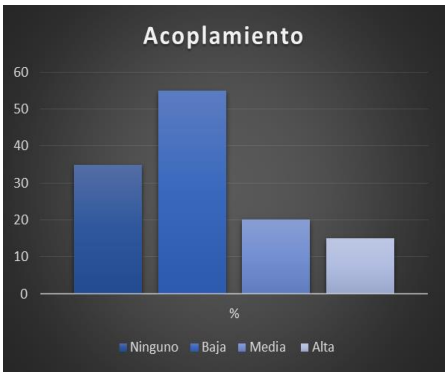
Resultados de la métrica TOC

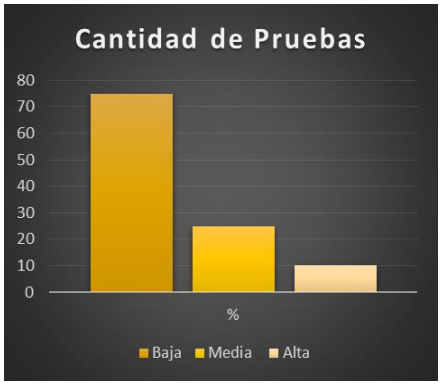
Representación en % de la incidencia de los resultados en los atributos de calidad:



Resultados de la métrica RC

Representación en % de la incidencia de los resultados en los atributos de calidad:





Anexo 4: Resultados de las pruebas con el framework JUnit

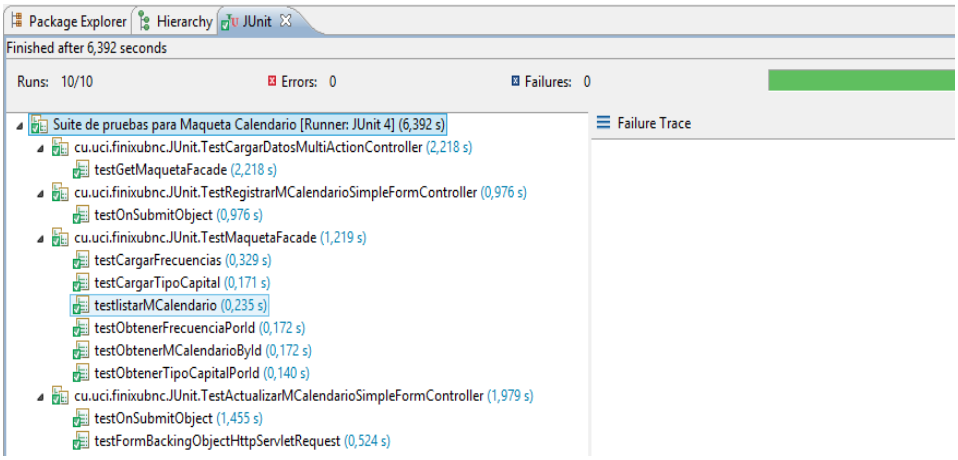


Figura 23: Suite de pruebas para el Módulo Gestionar Maquetas

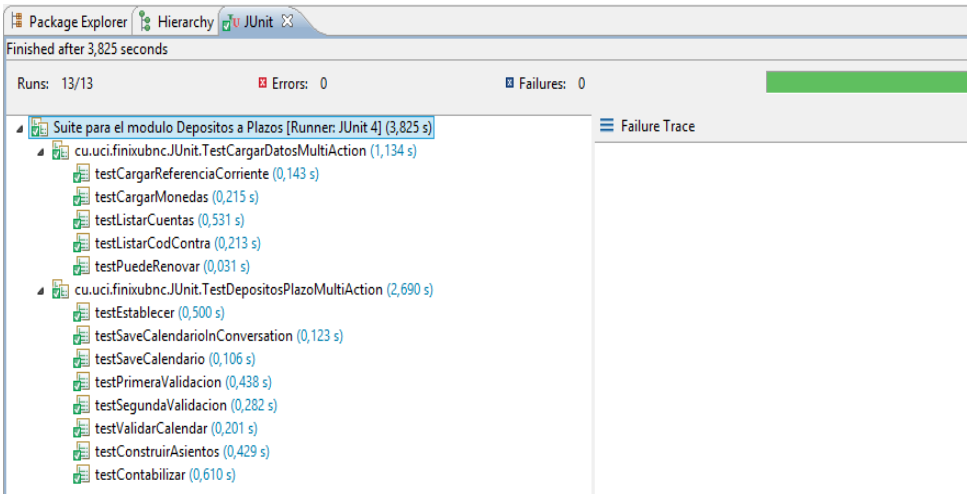


Figura 24: Suite de pruebas para el Subsistema Depósitos

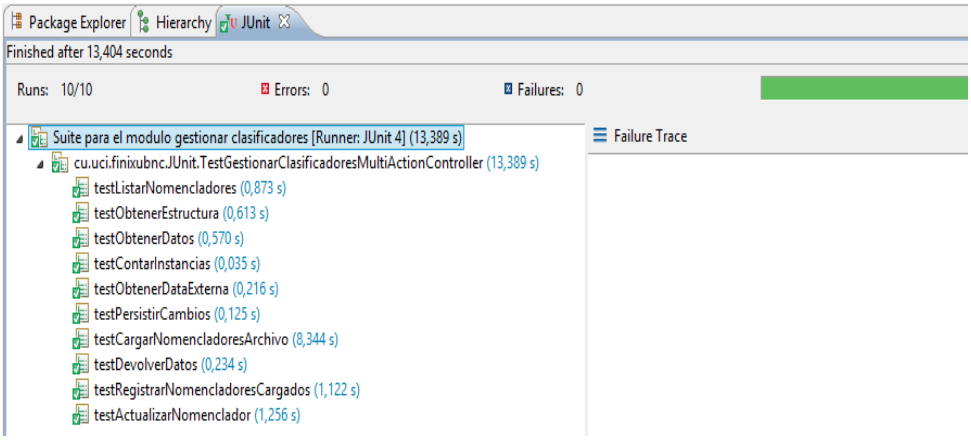


Figura 25: Suite de pruebas para el Módulo Gestionar Nomencladores

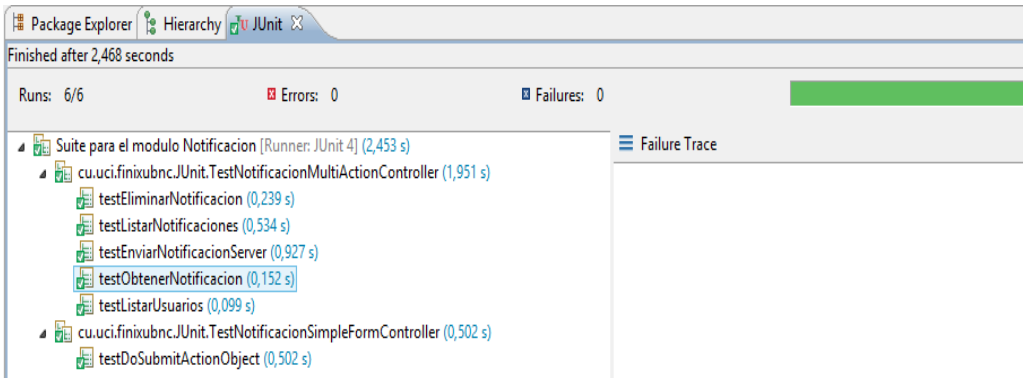


Figura 26: Suite de pruebas para el Módulo Notificaciones

Anexo 5: Casos de Prueba

Los Casos de Prueba aplicados al resto de los requisitos se encuentran en la carpeta anexada al documento: *Casos de Prueba*. El mismo contiene los siguientes informes:

Módulo Gestionar Maquetas:

- CIG-QF2-N-VEN-i5101: RF - Crear Maqueta de Calendario
- CIG-QF2-N-VEN-i5102: RF - Registrar Maqueta de Calendario
- CIG-QF2-N-VEN-i5103: RF - Cargar Maqueta de Calendario
- CIG-QF2-N-VEN-i5104: RF - Actualizar Maqueta de Calendario
- CIG-QF2-N-VEN-i5105: RF - Buscar Maqueta de Calendario
- CIG-QF2-N-VEN-i5106: RF - Eliminar Maqueta de Calendario

Subsistema Depósitos:

- CIG-QF2-N-DEP-i5101: RF - Renovar depósito a plazo

Módulo Gestionar Nomencladores:

- CIG-QF2-N-CON-i5101: RF - Cargar Tabla de Proveedores e Instituciones
- CIG-QF2-N-CON-i5102: RF - Registrar Nomencladores

Módulo Notificaciones:

- CIG-QF2-N-ADM-i5101: RF - Enviar Notificaciones
- CIG-QF2-N-ADM-i5102: RF - Enviar Orden de Cerrado
- CIG-QF2-N-ADM-i5103: RF - Gestionar Mensajes Predeterminados