Universidad de las Ciencias Informáticas
Facultad 3

Desarrollo del subsistema Reportes de Quarxo

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

Autores:
Hector Peña Torres
Efraín García Díaz

Tutores:
Ing. Lissett Díaz Mesa
Ing. José Antonio Sánchez Imbert

La Habana, 7 de junio de 2012

"Año 54 de la Revolución"

"(...) parecía que habíamos llegado al final del camino y resulta que era sólo una curva abierta a otro paisaje y a nuevas curiosidades."

José Saramago

Frase tomada de la novela: El año de la muerte de Ricardo Reis.
Imagen (en marca de agua): "Camino de Otoño". Autor: Miguel Gil García.

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2012.

Hector Peña Torres	Efraín García Díaz
Ing. Lissett Díaz Mesa	Ing. José Antonio Sánchez Imbert

DEDICATORIA

Hector:

A mi abuela Daysi, que desde el cielo me cuida y me guía.

A mis padres Asela y Héctor, que son la razón de que hoy este aquí.

A mi esposa Chavelys, por ser tú, solamente tú, de quien un día me enamoré.

Efraín:

A mis padres.

A mis abuelos Mayra y Miguel, que son las personas más importante de mi vida.

AGRADECIMIENTOS

Hector:

Gracias infinitamente:

A mi abuela Daysi, por su educación, por su amor, su cariño, su ternura, su paciencia y porque siempre en mi corazón será lo más grande y bello que pueda recordar.

A mi mamá, por su amor, su confianza sin límites, por su entrega. Gracias por todo, por inspirarme y por darme fuerzas para seguir.

A mi padre, por ser mi meta, mi guía brillante. Por ser la respuesta a cada una de mis preguntas, gracias por estar.

A mis hermanos, Javier (mi segundo padre), Michel y Daysita, porque de cada uno llevo algo más que sangre en mi corazón, por el cariño que me brindan, los quiero y los admiro.

A mi esposa, por su amor infinito, por su paciencia, por llenar mi vida de momentos tan especiales y bellos, por comprenderme y aceptarme sin reservas. Te amo “de aquí hasta allí”.

A mi familia, porque de una forma u otra, siempre están a mi lado.

A mi suegra, por su cariño, por aceptarme en su familia incondicionalmente, por su ayuda en todos estos años.

A los verdaderos amigos, los que no necesitan nombres, gracias por levantarme en mis tropiezos, por quererme incondicionalmente y permitirme entrar en sus vidas para siempre.

A mis compañeros de estos cinco años, a quienes quiero mucho y nunca voy a olvidar, gracias por los momentos de alegría, por enseñarme, por compartir esta etapa inolvidable.

A todos los que han contribuido en mi formación profesional y en la culminación de esta investigación.

A todos, gracias.

Efraín:

A mis padres que son la razón por la que me he esforzado todos estos años. Son las personas más importantes de mi vida y sin el amor y el cariño de ellos no hubiese alcanzado todas las metas que me propuse, incluyendo esta.

A mis abuelitos queridos Mayra y Miguel, que son uno de los tesoros más bellos que tengo.

Al resto de mi familia por apoyarme de una forma u otra en el trascurso de mi vida como estudiante.

A mi novia y futura esposa Daileny, por estar a mi lado todos estos años y por darme el ánimo que necesité en los buenos y malos momentos que he pasado en esta universidad.

A mis suegros Elba y Julio, que siempre se preocuparon por mí y por mi familia.

A mi compañero de tesis Héctor, que sin su esfuerzo y dedicación no hubiésemos salido adelante.

A todos mis amigos que durante estos 5 años han estado apoyándome.

A Yanko y Susy.

A todos muchas gracias.

RESUMEN

En la actualidad, el uso de las tecnologías informáticas es imprescindible para agilizar los procesos en las empresas. Constantemente se crean y perfeccionan herramientas que contribuyen a alcanzar mejores tiempos de respuesta y eficiencia ante necesidades cotidianas, como son los generadores de reportes. En el Banco Nacional de Cuba los reportes constituyen elementos esenciales en la toma de decisiones de los directivos en las distintas instancias, pues representan los datos relevantes de manera estructurada y resumida. Sin embargo, el sistema utilizado actualmente no cuenta con una interfaz gráfica amigable, por lo que crear, modificar y visualizar los reportes se hace muy complejo. Igualmente, no posibilita filtrar los datos según valores definidos por el usuario implicando el rediseño de todo el informe. El presente trabajo de diploma pretende contribuir a erradicar las dificultades mencionadas mediante el desarrollo del subsistema Reportes de Quarxo. Para un mayor entendimiento del tema de la gestión de reportes, se hace referencia a los principales conceptos utilizados en la investigación. Se presentan las principales características consideradas en el desarrollo de la aplicación, obtenidas en el estudio de la generación de reportes en sistemas de gestión nacional e internacional. Además, se describen las principales tecnologías utilizadas y la metodología de desarrollo del software, así como los elementos fundamentales de la arquitectura, análisis, diseño, implementación y los resultados del proceso de pruebas.

Palabras claves: ahorro, gestión, Quarxo, reportes.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las tecnologías de la información y las comunicaciones se han ido insertando velozmente en todas las actividades del quehacer humano, revolucionando la forma de pensar y actuar de las personas. Su uso se torna casi imprescindible para estar acorde al acelerado desarrollo en todos los sectores de la sociedad actual, haciendo especial énfasis en el empresarial.

El avance creciente en las tecnologías informáticas provee hoy a las empresas de sistemas cada vez más eficientes, que agilizan la gestión de los procesos administrativos. Han sustituido lo que normalmente consumía mucho tiempo y recursos en tareas engorrosas y repetitivas, por rapidez y precisión en el manejo de mayor volumen de información. De este modo, los diferentes sistemas no solo amplían los niveles de productividad y competitividad de las empresas, sino que a través de las nuevas tecnologías pueden tener un mayor alcance de sus recursos, y contar con niveles de información más precisos, útiles y actualizados.

Un ejemplo fehaciente del uso de estas tecnologías es el constante perfeccionamiento de los generadores de reportes, herramientas imprescindibles en los sistemas informáticos actuales. Un reporte puede considerarse un documento que representa datos relevantes de manera estructurada y resumida, a fin de transmitir información. En el ámbito de la Informática, los reportes son informes que organizan y exhiben los datos contenidos en una base de datos, con un diseño atractivo y fácil de interpretar por los usuarios. Esto contribuye en gran medida a la toma de mejores decisiones y permite a su vez conocer cuán cerca o lejos se está de alcanzar las metas u objetivos trazados.

Hoy en día la gestión de reportes en las entidades financieras bancarias forma parte de sus procesos fundamentales. A través de los reportes los bancos pueden realizar estimaciones acerca de sus estados financieros, así como realizar estudios y balances sobre la situación actual de sus cuentas, monedas, créditos y carteras. Dentro del Sistema Bancario Cubano constituyen elementos primordiales en la toma de decisiones de los directivos en las distintas instancias, ya que se emiten de manera cotidiana en todas las áreas de la entidad.

El sistema que está en explotación actualmente en el Banco Nacional de Cuba (BNC), es el Sistema Automatizado para la Banca Internacional de Comercio (SABIC) en su versión para MS-DOS. Hasta la fecha ha contribuido a agilizar los procesos y gestionar toda la información dentro de la institución. Sin embargo, a pesar de los innegables beneficios de su uso, este sistema se ha quedado por detrás ante los cambios tecnológicos que han ido sucediendo en los últimos 30 años.

El SABIC tiene como principal desventaja el ser monotarea, lo que entorpece en gran medida el trabajo del operador. Este sistema no permite generar reportes según valores definidos por el usuario para parámetros que posibiliten filtrar la información, para ello es obligatorio el rediseño del reporte.

Cuando la institución requiere un nuevo reporte o modificar uno existente los especialistas tienen que usar comandos de texto para su diseño e implementación, tarea complicada teniendo en cuenta que por lo general son extensos, complejos y hay que revisar todo el código programado para saber dónde hay que realizar los cambios pertinentes. Esto provoca que los tiempos de respuesta a estas solicitudes tarden más de lo debido, atrasándose cualquier tipo de actividad que dependa directamente de la emisión del reporte en cuestión. No existe una interfaz donde visualizar una vista preliminar de lo que se va diseñando por lo que todo el diseño se hace a código o a simple vista.

Las demoras provocadas por la tecnología en la elaboración de los reportes, así como la inexistencia de una interfaz dirigida al especialista encargado de esta tarea, dificultan la toma de decisiones importantes por parte de los directivos del BNC.

Cuando se emite un reporte se guarda en un archivo de texto (TXT) que solo es visible a través del sistema. Esto implica que los especialistas dependan del SABIC para consultar los resultados obtenidos, ya que la versión MS-DOS utilizada no soporta formatos de textos convencionales tales como PDF, EXCEL, HTML, etc. La portabilidad de los reportes se reduce a tener que imprimirlos en papel, incrementando el consumo de recursos materiales y disminuyendo la productividad de los trabajadores, ya que visualizarlo en la pantalla del sistema restringe al operador a trabajar únicamente con ese reporte y no se logra una vista detallada del mismo.

Con la modernización del Sistema Bancario Cubano, el BNC inició un proyecto de colaboración con el Centro de Informatización y Gestión de Entidades (CEIGE), de la

Universidad de las Ciencias Informáticas (UCI) para desarrollar el sistema Quarxo. Con esta solución se pretende agilizar y mejorar todos los procesos que se desarrollan en la institución. En su concepción se incluye igualmente la gestión de los reportes que responden a las diferentes áreas del BNC, aspecto en el cual se identifican las dificultades que han sido descritas anteriormente.

Se identificó como **problema de la investigación**: ¿Cómo apoyar la gestión de los reportes en el BNC para disminuir los tiempos de respuesta y ahorrar recursos materiales?

El **objeto de estudio** se enmarca en: la gestión de reportes delimitando como **campo de acción**: la gestión de reportes en el BNC.

El **objetivo general** es: Desarrollar el subsistema Reportes de Quarxo para contribuir a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

Para dar solución al problema descrito, se planteó como **idea a defender**: “Si se desarrolla el subsistema Reportes de Quarxo, se contribuirá a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución”.

De acuerdo con lo anterior se declaran los **objetivos específicos** siguientes:

1. Fundamentar la investigación mediante la elaboración del marco teórico.
2. Obtener los requisitos funcionales y no funcionales para el subsistema de Reportes de Quarxo.
3. Analizar la solución del subsistema Reportes de Quarxo.
4. Diseñar la solución del subsistema Reportes de Quarxo.
5. Implementar y validar el subsistema Reportes.

Para darle cumplimiento a los objetivos se trazan las siguientes **tareas**:

- ✓ Caracterización de los procesos relacionados con la generación y consulta de reportes en las entidades financieras bancarias.
- ✓ Valoración de las herramientas existentes para la generación de reportes.
- ✓ Caracterización de las tecnologías y herramientas a utilizar en el desarrollo del subsistema Reportes.

- ✓ Identificación, especificación y validación de los requisitos funcionales del software.
- ✓ Elaboración de los diagramas de clases del análisis y diagramas de colaboración correspondientes al análisis del subsistema Reportes.
- ✓ Descripción de los elementos arquitectónicos correspondientes al análisis del subsistema Reportes.
- ✓ Elaboración de los diagramas de clases del diseño, diagramas de secuencia y diagrama de despliegue correspondientes al diseño del subsistema Reportes.
- ✓ Caracterización e identificación de los patrones utilizados en la construcción del subsistema Reportes.
- ✓ Confección del modelo de componentes correspondientes a la implementación del subsistema Reportes.
- ✓ Implementación del subsistema Reportes.
- ✓ Validación mediante la aplicación de pruebas de caja blanca, pruebas de caja negra y pruebas de aceptación del cliente al subsistema Reportes.

El presente documento cuenta con un resumen, un índice de contenidos así como otros de figuras y tablas respectivamente. Presenta una introducción, tres capítulos, seguido de conclusiones, recomendaciones, una sección dedicada tanto a la bibliografía consultada como a las referencias bibliográficas y un glosario de términos.

Capítulo 1 Enfoques investigativos sobre la gestión de reportes: Se presentan los elementos teóricos que sirven de base a la investigación del problema planteado. Se realiza un estudio del estado del arte de las herramientas que se utilizan para la gestión de reportes. Además, se hace referencia a las diferentes tecnologías seleccionadas para el desarrollo de la investigación.

Capítulo 2 Análisis y diseño del subsistema Reportes de Quarxo: Se analiza y se diseña la solución propuesta mediante la descripción y modelación de los principales artefactos de los flujos de trabajo correspondientes. Se explica la utilización de los distintos patrones utilizados.

Capítulo 3 Implementación y validación del subsistema Reportes de Quarxo: Se explican los aspectos fundamentales de la implementación, modelo de componentes, nomenclatura, y se describe la arquitectura. Además, se realizan al software, las pruebas de caja blanca y pruebas de caja negra.

CAPÍTULO 1: ENFOQUES INVESTIGATIVOS SOBRE LA GESTIÓN DE REPORTES

Introducción

En este capítulo se abordan aspectos fundamentales que constituyen el soporte teórico para el desarrollo de la investigación. Se analizan algunos sistemas existentes que tienen integradas herramientas para la generación de reportes, a fin de seleccionar características relevantes que puedan resultar necesarias al BNC en cuanto a la gestión de reportes. Se estudia el estado del arte de algunas herramientas y tecnologías muy usadas actualmente para el diseño y creación de reportes. Además, se describen las características de la metodología de desarrollo, lenguaje de programación, entorno de desarrollo integrado, patrones, *frameworks* y otros elementos propuestos en el Plan de Desarrollo de Software del Sistema Automatizado de Gestión Bancaria para lograr un desarrollo exitoso del subsistema.

1.1. Conceptos fundamentales

En esta sección se explican algunos aspectos fundamentales para la comprensión del contexto de la investigación.

1.1.1. Reportes

Un reporte es un documento, una noticia o informe que brinda información con algún propósito y que presenta de manera estructurada y/o resumida, datos relevantes acerca de uno o varios temas en específico.

Un reporte combina tres tipos de información:

- ✓ Datos y su estructura.
- ✓ Diseño o información de formato que describe cómo serán presentados los datos.
- ✓ Propiedades del reporte, como autor, parámetros, imágenes dentro del reporte, entre otras. [1]

1.1.2. Reportes financieros

Los reportes financieros deben reflejar realmente las actividades de la compañía como base para la toma de decisiones. Debe observarse de manera cuidadosa la

divulgación informativa. Los reportes abarcan las aseveraciones relevantes de los Estados Financieros, en aspectos como: Existencia, Totalidad, Derechos y Obligaciones, Valuación, Presentación y Divulgación. Es importante tener en cuenta el criterio de la Materialidad (Valor Financiero Significativo). Estos reportes son de vital trascendencia ya que permite satisfacer necesidades de información que son requeridas para la toma de decisiones. [2]

Los reportes financieros son una poderosa herramienta para determinar cómo las empresas manejan sus responsabilidades sociales dentro de un ambiente común. La realización de Balances Contables, Balances Globales y Balances Generales permite medir el impacto a corto o largo plazo de la actividad de una empresa bancaria.

1.1.3. Formatos más utilizados para visualizar un reporte

Existe una gran variedad de formatos para exportar o visualizar los reportes, documentos y archivos que contengan información tabulada. Se diferencian entre ellos por el nivel de abstracción para mostrar los datos con la calidad requerida, por la adaptación a los distintos sistemas, y por el fácil manejo y entendimiento de los mismos. Entre los formatos más importantes se destacan:

PDF, por sus siglas en inglés *Portable Document Format*, formato de documento portátil, formato de almacenamiento de documentos. Desarrollado por la empresa *Adobe Systems*. Es de tipo compuesto (imagen vectorial, mapa de *bits* y texto).

XML, por sus siglas en inglés *eXtensible Markup Language*, lenguaje de marcas extensible. Es un metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos.

HTML, por sus siglas en inglés *HyperText Markup Language*, lenguaje de marcado de hipertexto. Es el lenguaje de marcado predominante para la elaboración de páginas web.

CVS, por sus siglas en inglés *Concurrent Versioning System*, es una aplicación informática que implementa un sistema de control de versiones. Mantiene el registro de todo el trabajo y los cambios en los ficheros que forman un proyecto y permite que distintos desarrolladores colaboren.

RTF por sus siglas en inglés *Rich Text Format*, formato de texto enriquecido. Es un formato de archivo informático desarrollado por *Microsoft* en 1987 para el intercambio de documentos multiplataforma.

XLS. *Microsoft Excel* es una aplicación para manejar hojas de cálculo. Este programa es desarrollado y distribuido por *Microsoft*, y es utilizado normalmente en tareas financieras y contables.

TXT es un archivo de texto llano o texto plano (*Plain Text).* Es un archivo informático compuesto únicamente por texto sin formato, solo caracteres, lo que lo hace también legible por todas las personas.

DBF es el acrónimo de *DataBase File*, el archivo es ampliamente utilizado en muchas aplicaciones que necesitan un formato simple para almacenar datos estructurados.

1.2. Mecanismos para la obtención de reportes

En cuanto al desarrollo de mecanismos para la obtención de reportes se pueden identificar dos tendencias: el desarrollo de soluciones propias ajustadas al negocio en cuestión y el uso de herramientas especializadas. [3]

A continuación se analiza la generación de reportes desarrollada en algunos sistemas informáticos, basada en la primera de las tendencias expuestas. A pesar de que en ninguno de los casos se ajusta al negocio del BNC, se pretende identificar algunas funcionalidades a considerar en la construcción del subsistema Reportes. Luego se presentan algunas de las principales herramientas especializadas en la gestión de reportes en el mundo, en función de la segunda tendencia.

1.2.1. Soluciones ajustadas al negocio

Sistema de Reportes en Línea (SIREL)

Es un sistema de información en ambiente web, desarrollado por la Unidad de Información y Análisis Financiero (UIAF) de la República de Colombia como mecanismo principal para recibir los reportes de información en línea. A través de este las entidades que reportan información a la UIAF pueden diligenciar y/o cargar completamente en línea sus informes de forma eficiente y segura. Muestra un reporte en una página HTML dando la posibilidad de imprimir. [4]

Sistema de Administración Financiera

Es un sistema creado en Ecuador para fomentar la investigación y estudio sobre todos los aspectos de la economía y la gestión empresarial. Este cuenta con un módulo de reportes dinámicos de ingresos y de gastos durante la ejecución del presupuesto de la organización. Le permite al usuario seleccionar datos específicos a través de filtros con respecto al resto de la información que se encuentra disponible. No tiene limitantes en el número de filtros que puede utilizar. Si bien es cierto los valores de filtrado son opcionales, pero constituyen el principal parámetro para obtener un reporte adecuado. [5]

VERSAT-Sarasola

Es un sistema económico integrado, constituido por diez módulos o subsistemas, que incluye un generador de reportes conocido como “Complementos del VERSAT”. Esta herramienta permite obtener información sin mayores complejidades de uso, expresando sus resultados en una gama de funciones tipo y predeterminadas. Interactúa con *Microsoft Excel*, utilizando sus facilidades para el diseño de información. La concepción de estas funciones está ligada a los distintos subsistemas del VERSAT Sarasola. [6]

Rodas XXI: Solución integral

Es un sistema integral económico administrativo cubano que posibilita automatizar el funcionamiento de cualquier empresa o unidad presupuestada nacional. En la actualidad es empleado por más de 450 entidades del país. Crea reportes sin complejidad, permite la visualización de los mismos y, de forma opcional los imprime. Exporta las bases de datos de los módulos a ficheros con estructura DBF o TXT, y lleva un registro de las operaciones relacionadas con el sistema, que permite auditar el mismo. [7]

En conclusión, la gestión de reportes en ninguno de los casos anteriores se ajusta al negocio del BNC. Sin embargo, con su estudio se identificaron funcionalidades relevantes a considerar en la construcción del subsistema Reportes, tales como: la **creación**, **visualización**, **impresión** y **exportación** de los reportes.

1.2.2. Herramientas especializadas

Crystal Report

Crystal Report es la herramienta de elaboración de informes estándar para *Microsoft Visual Studio*. Permite crear contenido interactivo con calidad de presentación en dicha plataforma. Incluye filtros de exportación a los formatos *Adobe PDF*, *Microsoft Excel*, *Microsoft Word*, RTF, HTML y DHTML. Permite agregar datos de otras fuentes, filtrar datos, agruparlos y ordenarlos en aras de mejorar la presentación del informe. Posee como principales desventajas:

- ✓ Es propietario.
- ✓ Las licencias son muy costosas.
- ✓ Para modificar el reporte es preciso recompilar el proyecto donde se encuentra lo que afecta la portabilidad de los reportes. [8]

ActiveReports

Está escrito completamente en C# y ofrece integración con *Microsoft Visual Studio*. Se licencia por desarrollador y es de libre distribución. El producto incluye un asistente de informes y un asistente de conversión de informes de *Microsoft Access* para trabajar rápidamente. *ActiveReports* incluye filtros de exportación a los formatos *Adobe PDF*, *Microsoft Excel*, RTF, HTML, Texto y TIFF. También tiene un visor *Windows Viewer* con soporte para la visualización simultánea de múltiples páginas, un panel de Tabla de Contenidos con una nueva pestaña de miniaturas, posibilidad de búsqueda de informes y personalización completa de su barra de herramientas. Sus principales desventajas son:

- ✓ Es propietario.
- ✓ Las licencias son costosas. [9]

Reporting Service

Es una potente herramienta para generar reportes a partir de servidores de *Microsoft SQL Server*. Utiliza el formato estándar XML e integrada en todas las versiones. Puede crear reportes interactivos, tabulares o de forma libre que recuperen datos a intervalos programados o a petición, cuando el usuario abra un reporte. Se puede elegir entre diversos formatos de visualización para representar los informes a petición en los formatos preferidos para la manipulación o impresión de datos tales como (PDF, XML, HTML, etc.). Además, se pueden utilizar reportes con parámetros para filtrar datos basados en valores que se proporcionan en tiempo de ejecución. Las principales desventajas son:

- ✓ Es propietario.
- ✓ Dependiente de la plataforma .NET.
- ✓ Fuente de datos restringida a servidores de *Microsoft SQL Server*. [10]

DataVision

Es una herramienta de reporte de código abierto similar a *Crystal Report*. Los reportes son diseñados en una interfaz gráfica bastante sencilla e intuitiva. Permite visualizar e imprimir desde la aplicación o exportar como HTML, XML, PDF, Excel o archivos de texto delimitados por tabuladores o comas. *DataVision* está escrito en Java y puede generar informes de bases de datos o archivos de datos de texto. Funciona con cualquier base de datos que tenga un dispositivo de software JDBC: Oracle, PostgreSQL, MySQL, HSQLDB y *Microsoft Access*. Presenta como desventajas:

- ✓ Dependiente del origen de datos.
- ✓ Poca portabilidad de los reportes. [11]

Generador Dinámico de Reportes (GDR) del sistema PATDSI

PATDSI es un paquete de herramientas para la ayuda a la toma de decisiones. El mismo es concebido como una plataforma Web única de inteligencia de negocio, que integra en sí las funcionalidades más recurrentes y específicas necesarias para la toma de decisiones en diferentes contextos. Fue creado por el Centro de Tecnologías de Gestión de Datos (DATEC), perteneciente a la UCI.

Características:

- ✓ Un diseñador de modelo semántico que permite al usuario conectarse a la fuente de datos especificada y definir lo que necesita para construir los reportes. Esta fuente de datos pueden ser tablas, consultas o funciones de una base de datos en gestores como PostgreSQL, ORACLE, entre otros.
- ✓ Un diseñador de consulta que ayuda a un usuario inexperto en lenguaje SQL a construir visualmente la consulta con los datos que necesita para mostrar en el reporte.
- ✓ Un diseñador de reportes con una alta usabilidad que garantiza construir visualmente las plantillas de los reportes que se deseen.
- ✓ Un visor de reportes que permite mostrar al usuario la salida del reporte diseñado y de esta forma exportarlo a múltiples formatos como PDF, HTML, CVS y XML.

- ✓ Un administrador de reportes que está dirigido hacia usuarios más avanzados del sistema, los cuales tienen la posibilidad de configurar las categorías de los reportes, los permisos para el acceso a los mismos y las vías de entrega de dichos reportes, las cuales incluyen suscripciones por correo electrónico y por FTP.

***Requisitos*:**

1. Un servidor Apache (Recomendable 1GB de RAM).
2. Un servidor de base de datos para PostgreSQL (Recomendable 1GB de RAM).
3. En el cliente un navegador web (Recomendable Mozilla Firefox). [12]

IReport

IReport (IR) es un constructor/diseñador de informes desarrollado en Java. Permite a los desarrolladores crear reportes visualmente a través de una herramienta con una interfaz de usuario muy completa y fácil de usar. Provee las funciones más importantes para crear complejos informes con cartas, imágenes y subinformes con facilidad, lo que ahorra mucho tiempo. IR está además integrado con *JFreeChart*, una de las bibliotecas gráficas de código abierto más difundida para Java. Utiliza la librería *JasperReports* para todo el trabajo con los reportes constituyendo en ese sentido, el núcleo de la herramienta. Este instrumento permite que los usuarios corrijan visualmente informes complejos. Los datos para imprimir pueden ser recuperados por varios caminos incluso múltiples uniones JDBC, *TableModels*, *JavaBeans*, XML, entre otros. [13]

Características:

- ✓ 100 % escrito en Java y además de código abierto y gratuito. Maneja el 98 % de las etiquetas de *JasperReports*.
- ✓ Permite diseñar con sus propias herramientas: rectángulos, líneas, elipses, campos texto, cartas, y subreportes.
- ✓ Soporta internacionalización nativamente.
- ✓ Navegador de la estructura del documento.
- ✓ Recopilador y exportador integrados. Soporta JDBC.
- ✓ Soporta *JavaBeans* como orígenes de datos (estos deben implementar la interfaz *JRDataSource*). Incluye asistentes para crear automáticamente informes.
- ✓ Tiene asistentes para generar los subreportes.

- ✓ Tiene asistentes para las plantillas y posee gran facilidad de instalación.

1.2.2.1. Selección de la herramienta especializada

Es importante destacar que aunque la herramienta desarrollada en la UCI (Generador dinámico de reportes) es muy poderosa y posee potentes funcionalidades y facilidades, no se corresponde con las necesidades identificadas para el desarrollo del sistema. En la siguiente tabla se relacionan algunos aspectos que corroboran lo antes mencionado.

PARÁMETROS	PATDSI 2.0	IR	OBSERVACIONES
Consultas sobre tablas	Sí	Sí	
Consultas sobre Procedimientos Almacenados	No	Sí	La alternativa para este problema podrían ser las funciones.
Consultas sobre funciones	Sí	Sí	*SQLServer* no permite la ejecución del "*INSERT EXEC*".
Dependencias tecnológicas	Sí	No	*PostgreSQL*, *Apache*.
Integración	Sí (*)	Sí	(*) Utiliza peticiones http para obtener metadato y así gestionar los reportes desde la aplicación.

Tabla 1. PATDSI Vs IReport

En conclusión, de las herramientas especializadas en la generación de reportes estudiadas, se considera *IReport* para la construcción del subsistema Reportes, pues permite alcanzar los resultados esperados, a partir de sus características:

- ✓ Es la herramienta que mejor se integra con el *framework* que maneja el negocio de la aplicación.
- ✓ Es una herramienta completamente escrita en código abierto, posibilitando el acceso a su implementación.
- ✓ Constituye una solución concreta y confiable para facilitar la generación y la impresión de los reportes de una manera más eficiente, cubriendo las necesidades del cliente.
- ✓ Posee una amplia documentación y una gran variedad de ejemplos que sirven para ilustrar las potencialidades del mismo, además de ayudar en la productividad al usarlo.

1.3. Principales librerías de la herramienta IReport

Constituyen una forma de acelerar los tiempos de programación, bajar costos y agregar funcionalidades utilizando soluciones ya probadas. Consiste en una colección de programas, repositorios y paquetes de software que contienen fragmentos de código que se mantienen disponibles para ser reutilizados, transformados o consultados desde cualquier programa o subprograma. [14]

JasperReports

Es una librería de clases 100 % Java de código abierto desarrollada por *Teodor Danciu*[1]. Permite la creación, diseño, impresión y exportación de informes con la librería de Java más popular del mundo. Contiene técnicas para la integración con *Hibernate, Spring* y JSF por solo citar algunos, y posibilita exportar informes a diferentes formatos de archivo. Es fácil de integrar y los usuarios pueden configurar y redefinir las facilidades que brinda implementando algunas de sus interfaces. Los reportes generados con *JasperReports* pueden ser exportados a una multitud de formatos como PDF, XLS, RTF, HTML, XML, CVS y TXT. Manejar los reportes con *JasperReports* hace más cómodo el trabajo ya que son más fáciles de entender, más profesionales y de gran alcance a partir de datos desordenados y dispersos que utilizan una determinada fuente de datos. [15]

DynamicJasper

DynamicJasper (DJ) es una librería de código abierto que esconde la complejidad de *JasperReports*, ayudando a los desarrolladores a ahorrar tiempo cuando diseñan reportes de simple o mediana complejidad y generando la disposición de los elementos del reporte automáticamente. Permite definir programáticamente las columnas, grupos, totales, gráficos, subreportes y el formato de salida (PDF, EXCEL, HTML) en tiempo de ejecución. Su objetivo es abarcar el 99 % de los reportes que se basan en columnas dinámicas como también los que tienen grupos y cruces de datos. DJ posibilita agregar variables en las cabeceras y pie de las columnas y grupos con operaciones de suma, contar, etc. Se puede definir en tiempo de ejecución el orden de aparición de las columnas, los grupos, las variables, los estilos y demás propiedades propias del reporte. [16]

IText

[1] Fundador y arquitecto de JasperReports.

Es una biblioteca de código abierto escrita en Java que permite a los desarrolladores de una aplicación la creación y manipulación de archivos PDF. *IText* brinda muchas opciones como son servir un PDF a un navegador, generar documentos dinámicos desde ficheros XML o desde bases de datos, añadir marcadores o numerar un PDF. También dispone de un sinfín de opciones a la hora de diseñar los documentos, inserción de todo tipo de textos, tablas, listas, imágenes con distintos formatos, etc.[17]

La utilización de estas librerías contribuye al logro del objetivo principal de esta investigación debido a que poseen particularidades en sus funciones que se identifican con la arquitectura definida para la construcción del subsistema Reportes de Quarxo y que a continuación se explican.

1.4. Metodología de desarrollo de software

Se usará la metodología RUP (*Rational Unified Process*) por su facilidad de adaptación y configuración a proyectos con un período de duración considerable y por ser una metodología robusta y bien definida, probada anteriormente en otros proyectos con características similares a Quarxo. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Es un proceso de desarrollo de software, o sea, un conjunto de actividades para transformar los requisitos de un sistema de software.

La metodología en cuestión tiene entre sus principales características que está dirigida por casos de uso, es centrada en la arquitectura y además iterativo e incremental. Incluye artefactos, que son los productos tangibles del proceso, y roles, que no es más que el papel que desempeña una persona en un determinado momento. [18]

1.5. Lenguajes y herramientas de modelado

El lenguaje propuesto por la metodología de desarrollo RUP para modelar elementos de software es **UML** (*Unified Modeling Language*), el cual es un sistema de notación que se ha convertido en estándar en el mundo del desarrollo de sistemas. El mismo es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. [19]

Existen varios software para el modelado UML, son las denominadas herramientas CASE. Dichas herramientas son la aplicación de tecnologías informáticas a las actividades, técnicas y metodologías propias de desarrollo. Su objetivo es acelerar el proceso para el que han sido diseñadas, automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. La principal ventaja de la utilización de estas herramientas es la mejora de la calidad de los desarrollos realizados y el aumento de la productividad. La herramienta propuesta para el modelado de los procesos en esta investigación es el **Visual Paradigm**, la cual es muy eficaz para visualizar y diseñar elementos de software. VP está orientado a la creación de diseños usando el paradigma de programación orientada a objetos. Además ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite modelar diagramas de clases, código inverso y generar documentación. Provee soporte para la generación de código, tiene integración con diversos IDE como *NetBeans* (de *Sun Microsystems*), *JDeveloper* (de Oracle), Eclipse (de IBM), *JBuilder* (de *Borland*), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en Java, .NET, XML e *Hibernate*.

1.6. Arquitectura de software

Uno de los pasos fundamentales en el proceso de desarrollo de software es ciertamente la definición de la arquitectura del software, la cual constituye el diseño de más alto nivel de la organización de un sistema, programa o aplicación. Entre sus responsabilidades se destacan:

- ✓ Definir los módulos principales
- ✓ Definir las responsabilidades que tendrá cada uno de estos módulos
- ✓ Definir la interacción que existirá entre dichos módulos
- ✓ Control y flujo de datos
- ✓ Secuenciación de la información
- ✓ Protocolos de interacción y comunicación
- ✓ Ubicación en el hardware

La arquitectura del software aporta una visión abstracta de alto nivel, postergando el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La IEEE (*Institute of Electrical and Electronics Engineers*) define la arquitectura de software como: “La Arquitectura del Software es la organización fundamental de un

sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución". [20]

1.7. Patrones

1.7.1. Patrones de arquitectura

Expresan un esquema fundamental de organización estructural para sistemas de software. Proveen subsistemas predefinidos, especificando sus responsabilidades, e incluyen reglas y guías para organizar las relaciones entre ellos.

Modelo Vista Controlador (MVC)

El patrón MVC separa el modelo del dominio del negocio (clases de negocio), presentación y las acciones que se ejecutan sobre estas. El modelo administra el comportamiento y la información del modelo del negocio, de la misma forma que responde sobre acciones que se quieran efectuar sobre este. La vista, es la encargada de administrar la presentación de la información, y por último el controlador es el componente intermedio entre el modelo y la vista para lograr un desacoplamiento total entre los dos. [21]

1.7.2. Patrones de Diseño

Con el perfeccionamiento cada vez más creciente de los sistemas informáticos en el complejo mundo del desarrollo del software se ha hecho muy común el uso de los patrones. A continuación se describen algunos patrones de diseño a emplear en el desarrollo del subsistema Reportes, en el siguiente capítulo se analizarán las concurrencias de estos en el subsistema mediante su uso.

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

Según el libro "UML y Patrones" del autor *Craig Larman* publicado por la editorial *Prentice Hall* en el año 1999, se definen dos grupos de patrones:

Patrones GRASP

Los patrones generales para asignar responsabilidades del inglés GRASP (*General Responsibility Asignment Software Patterns*). Abarca cinco patrones principales: experto, creador, bajo acoplamiento, alta cohesión, controlador. Algunos de estos serán utilizados para contribuir a un mejor desarrollo del subsistema.

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

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

Patrón Bajo Acoplamiento: Pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir, 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.

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

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

Patrones GoF

Por sus siglas en inglés *Gang of Four*. Llamados así debido a que fueron cuatro autores los que escribieron el libro "*Design Patterns*" que ilustra sus funciones.

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

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

Patrón Cadena de Responsabilidad: 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.

Patrón Singleton: Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones. [19]

Patrón DAO

Por sus siglas en inglés *Data Access Object*. Este patrón maneja la conexión con la fuente de datos para obtener y almacenar datos. Su uso ofrece varios beneficios para la persistencia de datos como son: la separación el acceso a datos de la lógica de negocio, lo cual suele ser altamente recomendable en sistemas medianos o grandes, o que manejen lógica de negocio compleja. Permite el encapsulamiento de la fuente de datos, lo cual es especialmente beneficioso en sistemas con acceso a múltiples entradas; posibilita el ocultamiento de la API con la que se accede a los datos, lo que viabiliza que se pueda cambiar el negocio del manejo de los datos persistentes sin que afecte el resto, por ejemplo cambiar de *framework* de acceso a datos, centralizando todo el acceso a datos en una capa independiente. [22]

1.8. Ambiente de desarrollo

1.8.1. Lenguajes de programación

Entre **los lenguajes de programación del lado del servidor** más usados en el desarrollo web se encuentra PHP, Java y C#. El primero es un lenguaje gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. En cuanto a C# es un lenguaje orientado a objetos, desarrollado y estandarizado por *Microsoft* como parte de la plataforma .NET; y el lenguaje Java es fácil de usar, multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Dicho lenguaje además es compilado, utiliza el paradigma orientado a objeto y en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento. Es uno de los lenguajes más utilizados en la actualidad para el desarrollo de proyectos de software bajo la especificación de *Java Enterprise Edition* (JEE), el cual es una tecnología que apunta a simplificar el diseño y desarrollo de aplicaciones empresariales robustas, escalables y seguras utilizando Java como lenguaje. Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, ofreciendo un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja. [23]

En cuanto a **los lenguajes de programación del lado del cliente** se encuentra *JavaScript* el cual es compatible con la mayoría de los navegadores modernos, permite crear efectos especiales en las páginas web y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones *JavaScript* y ejecutarlas. Es un lenguaje bastante sencillo, rápido y fácil de aprender por personas de poca experiencia. Por otro lado existe XHTML que es el lenguaje de marcado pensado para sustituir el HTML. Se utiliza para generar documentos y contenidos de hipertexto generalmente publicados en la WEB. Es además una reformulación del lenguaje HTML que se puede jactar de ser ahora compatible con XML. [24]

1.8.2. Frameworks

Un *framework* es una estructura software compuesta de componentes personalizables e intercambiables que permiten desarrollar una aplicación. En otras palabras, un *framework* se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede agregar algunas piezas para construir una aplicación concreta. [25]

Se puede decir que un *framework* en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Son generadores de aplicaciones que se corresponden directamente con un dominio específico, es decir, con un grupo de problemas relacionados.

1.8.2.1. Spring Framework

Está compuesto por un conjunto de módulos, de los cuales se pueden tomar aquellos que faciliten la implementación del trabajo en cuestión. Ideado principalmente por Rob Johnson, es libre y de código abierto desde febrero de 2003. El centro de *Spring* está basado en el principio de Inyección de Dependencias. Esta técnica hace externa la creación y el manejo de las dependencias de las clases, con lo que se logra una mayor limpieza y claridad en el código. [26]

Es un *framework* que ha venido a revolucionar la manera de programar aplicaciones Java debido a la facilidad de crear componentes reutilizables. Además de que se adapta fácilmente con otros *frameworks* ofrece autonomías a los desarrolladores y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto haciendo uso de las prácticas comunes en la industria de software.

Spring contiene una gran variedad de características que le proporcionan una funcionalidad muy basta; dichas características están organizadas en siete módulos:

Spring Core: “Núcleo”, es la parte fundamental del *framework* ya que provee toda la funcionalidad de Inyección de Dependencias permitiendo administrar la funcionalidad del contenedor de *beans*.

Spring Context: “Contexto”, provee de herramientas para acceder a los *beans* de una manera elegante, similar a un registro JNDI. El paquete de contexto hereda sus características del paquete de *beans* y añade soporte para mensajería de texto.

Spring DAO: provee una capa de abstracción de JDBC que elimina la necesidad de teclear código JDBC tedioso y redundante. Permite que la programación del acceso a datos sea un poco más limpia y entendible. Permite administrar transacciones tanto declarativas como programáticas.

Spring ORM: provee capas de integración para API de mapeo objeto - relacional, incluyendo *Hibernate* e *iBatis*. El mapeo de ORM se puede usar en conjunto con otras

características que *Spring* como la administración de transacciones mencionada con anterioridad.

Spring AOP: provee una implementación de programación orientada a aspectos. Sus funcionalidades permiten desacoplar el código de una manera limpia implementando funcionalidades que por lógica y claridad debería estar separada.

Spring Web: provee características básicas de integración orientado a la web y se encarga de crear el contexto para las aplicaciones web. Incluye soporte para una variedad de tareas como la subida de archivos y la vinculación de parámetros de las peticiones a objetos del negocio.

Spring Web MVC: provee una implementación Modelo-Vista-Controlador para las aplicaciones web. La implementación de *Spring MVC* permite una separación entre código de modelo de dominio y las formas web.

1.8.2.2. Hibernate

Hibernate pertenece a los denominados *framework* ORM (Mapeo de Objetos Relacional) más populares y robustos del mundo. *Hibernate* propone un lenguaje de consulta orientado a objeto, HQL, el cual puede ejecutar cualquier tipo de consultas sin importar el gestor de base de datos.

Características de Hibernate:

- ✓ Permite expresar consultas utilizando SQL nativo o consultas basadas en criterios.
- ✓ Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones JEE y contenedores web, y por supuesto también puede utilizarse en aplicaciones de escritorio.
- ✓ Puede operar proporcionando persistencia de una manera transparente para el desarrollador.
- ✓ Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y el *framework* de colecciones de Java.
- ✓ Soporte para modelos de objetos con una granularidad muy fina. Permite una gran variedad de mapeos para colecciones y objetos dependientes.
- ✓ Provee un sistema de caché de dos niveles y puede ser usado en un *cluster*. Permite inicialización *lazy* (perezosa) de objetos y colecciones.

- ✓ Proporciona el lenguaje HQL el cual provee una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- ✓ Presenta un potente mecanismo de transacciones de aplicación llegando incluso a permitir la interacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).
- ✓ Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.
- ✓ Brinda una jerarquía detallada para tratar todas las excepciones que puede ocurrir al fallar la interacción con la base de datos. [27]

1.8.2.3. Dojo Toolkit

Es un *framework* de *JavaScript* que brinda varios instrumentos como controles del interfaz de usuario, efectos, utilidades comunes, una API para gestionar el acceso asíncrono al servidor, etc.

Es una colección de *scripts* estáticos que permiten el desarrollo de aplicaciones web enriquecidas en el cliente, incorpora soporte para el trabajo con la tecnología AJAX. Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, hace transparente el desarrollo para diferentes implementaciones del DOM, ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS y *JavaScript* para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: dojo, dijit y dojox. [28]

1.8.3. Herramientas de desarrollo

Eclipse IDE

Eclipse es un IDE de código abierto y multiplataforma desarrollado por IBM. Emplea plugins para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura de plugins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo,

tales como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, entre otros. [29]

Se estará haciendo uso del Eclipse IDE GALILEO.

Contenedor Web (Tomcat)

Apache Tomcat es un software de código abierto implementado para las tecnologías *Java Servlet* y *Java Server Pages*. Es desarrollado en un entorno abierto, participativo y publicado bajo la licencia del software de *Apache*. Se ejecuta en varios sistemas operativos. Es un servidor configurable de diseño modular, con diversidad que permiten garantizar una elevada seguridad y buenas prestaciones. Además, brinda soporte para la plataforma de desarrollo JEE. Existe abundante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente. [30]

Control de Versiones (SubVersion)

SubVersion (SVN) es un software libre desarrollado para el control de versiones. Es un sistema centralizado para compartir información que permite realizar modificaciones atómicas y gestionar archivos, directorios y sus cambios a través del tiempo, lo que facilita las tareas administrativas. Su capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. [31]

Sistema Gestor de Base Datos

SQL Server 2005 es un servidor de base de datos basados en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración, permite además trabajar en modo cliente-servidor y promueve la escalabilidad y seguridad de la información. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés “*Transact-SQL*), siendo este último el principal medio de programación y administración del servidor. Requiere para su funcionamiento del sistema operativo *Windows*, representando esto un inconveniente para su utilización. [32]

Su uso está basado en una decisión del BNC.

1.9. Conclusiones del capítulo

- ✓ En el estudio de los mecanismos para la obtención de reportes, se descartó la utilización de las aplicaciones informáticas ajustadas al negocio, por no corresponderse con la solución que se necesita. Se consideró *IReport* como herramienta especializada para la generación de los reportes.
- ✓ Las tecnologías seleccionadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales: *JasperReports* como librería especializada en la generación de reportes, UML como lenguaje de modelado, *Visual Paradigm* como herramienta CASE, Java (en su especificación JEE) como lenguaje de programación, *Spring* como *framework* núcleo de la aplicación apoyado por los *frameworks Hibernate* y *Dojo Toolkit*, Eclipse como herramienta de desarrollo, *SQL Server 2005* como gestor de base de datos, y como contenedor web el *Apache Tomcat*.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SUBSISTEMA REPORTES DE QUARXO

Introducción

En este capítulo se presentan los primeros flujos de trabajo propuestos por RUP para el desarrollo de la solución propuesta: Requisitos, Análisis y Diseño. En la etapa de Requisitos se identifican y describen los requisitos funcionales y no funcionales, y se construye el Modelo de casos de uso del subsistema propuesto. En el Análisis y el Diseño, se definen todos los artefactos propuestos por RUP para cada caso, destacando el Modelo de análisis y Modelo de diseño, y la descripción de la arquitectura como base para el desarrollo de este tipo de aplicación. Igualmente, se especifican los patrones de arquitectura y de diseño utilizados en la concepción de la solución.

2.1. Requisitos

En este flujo de trabajo se analizan los aspectos correspondientes a las conciliaciones del cliente y los directivos del proyecto con relación a las funcionalidades que el sistema debe permitir y las que debe cumplir. El propósito fundamental de este flujo de trabajo es guiar el desarrollo hacia un sistema más completo. Esto se consigue mediante una descripción de los requisitos del sistema lo suficientemente acertada como para que pueda llegarse a un entendimiento mutuo entre el cliente y los desarrolladores sobre qué debe y qué no debe hacer el sistema. [33]

2.1.1. Comprensión del contexto del sistema. Modelo de dominio

Existen varias formas para describir o expresar el contexto de un sistema en una forma utilizable por los desarrolladores de software. Una de estas vías es mediante la construcción de un modelo de dominio.

Según el libro titulado “El Proceso Unificado De Desarrollo De Software”, de los autores Ivan Jacobson, Grady Booch y James Rumbaugh: este modelo describe los conceptos importantes del contexto como objetos del dominio, y enlaza estos objetos unos con otros. La identificación y asignación de un nombre para estos objetos ayuda a completar un glosario de términos que permitirá comunicarse mejor a todas aquellas personas que trabajan en el sistema.

El modelo de dominio puede representarse mediante UML, con un Diagrama de Clases en los que se muestran:

- ✓ Objetos del negocio que representan elementos que se manipulan en el negocio.
- ✓ Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento.
- ✓ Sucesos que ocurrirán o han ocurrido. [33]

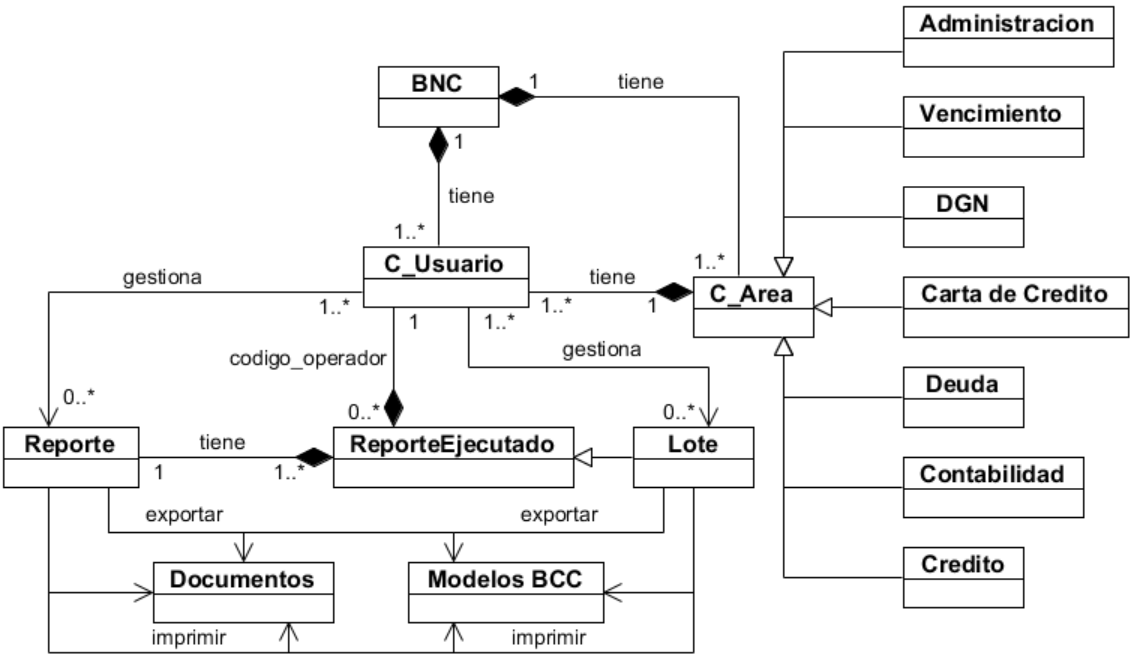


Figura 1. Modelo de dominio.

2.1.2. Especificación de requisitos funcionales

Los requisitos funcionales son los que definen el comportamiento interno del sistema. Abarcan las funcionalidades específicas que el software debe ser capaz de realizar. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. [20]

Requisitos funcionales (RF)

RF 01 Exportar reportes a PDF.

El sistema permitirá visualizar y exportar todos los reportes en formato PDF.

Prioridad: Alta

RF 02 Exportar reportes a DBF.

El sistema permitirá exportar todos los reportes a DBF.

Prioridad: Alta

RF 03 Exportar reportes a Excel.

El sistema permitirá visualizar y exportar todos los reportes en formato Excel.

Prioridad: Alta

RF 04 Imprimir el resultado de la búsqueda.

El sistema permitirá imprimir el resultado de la búsqueda realizada, generando así un informe que contenga una tabla con los resultados visualizados.

Prioridad: Alta

RF 05 Listar los reportes a consultar.

El sistema permitirá listar los reportes a consultar con los siguientes datos:

- ✓ Identificador de Lote.
- ✓ Título Lote
- ✓ Fecha Lote
- ✓ Operador

Prioridad: Alta

RF 06 Listar los reportes a consultar según el orden seleccionado.

El sistema permitirá listar los reportes a consultar según el orden seleccionado a través de los parámetros:

- ✓ Identificador de Lote (Orden: ascendente, descendente).
- ✓ Título Lote (Orden alfabético: ascendente, descendente).
- ✓ Fecha Lote (Orden: ascendente, descendente).
- ✓ Operador (Orden alfabético: ascendente, descendente).

Prioridad: Baja

RF 07 Buscar el reporte a consultar.

El sistema permitirá buscar el reporte a consultar a partir de los siguientes criterios de búsqueda:

- ✓ Identificador de Lote.
- ✓ Título Lote
- ✓ Operador

En estos casos, podrá filtrarse la búsqueda por los siguientes elementos:

- ✓ Empiece con
- ✓ Termine con
- ✓ Igual
- ✓ No igual
- ✓ Contiene
- ✓ No contiene
- ✓ Fecha Lote

Por los siguientes elementos:

- ✓ Igual
- ✓ No igual
- ✓ Mayor igual que
- ✓ Menor igual que
- ✓ Mayor que
- ✓ Menor que
- ✓ Entre

Prioridad: Alta

RF 08 Visualizar el reporte a consultar.

El sistema permitirá visualizar el reporte a consultar, seleccionado de la lista de reportes.

Prioridad: Alta

RF 09 Listar los reportes a ejecutar.

El sistema permitirá listar los reportes a ejecutar con los siguientes datos:

- ✓ Identificador de Reporte.

- ✓ Título

Prioridad: Alta

RF 10 Listar los reportes a ejecutar según el orden seleccionado.

El sistema permitirá listar los reportes a ejecutar según el orden seleccionado a través de los parámetros:

- ✓ Identificador de Reporte (Orden: ascendente, descendente).
- ✓ Título (Orden alfabético: ascendente, descendente).

Prioridad: Baja

RF 11 Buscar el reporte a ejecutar.

El sistema permitirá buscar el reporte a ejecutar a partir de los siguientes criterios de búsqueda:

- ✓ Identificador de Reporte.
- ✓ Título

En ambos casos, podrá filtrarse la búsqueda por los siguientes elementos:

- ✓ Empiece con
- ✓ Termine con
- ✓ Igual
- ✓ No igual
- ✓ Contiene
- ✓ No contiene

Prioridad: Alta

RF 12 Ejecutar reporte

Este requisito funcional se divide en un total de 96 requisitos debido a que cada reporte constituye un caso de uso (CU) diferente lo que conlleva a un análisis, un diseño y una implementación diferente para cada reporte. A continuación se describe uno de los reportes: “Ejecutar Reporte Deuda Rusia”. Las restantes descripciones se pueden consultar en el documento de tesis en formato digital (*Anexo # 1*).

Ejecutar el reporte Deuda Rusia

El sistema permitirá ejecutar el reporte Deuda Rusia, a partir de los siguientes parámetros:

- ✓ Código del Banco (número de 4 dígitos que representa el código de cada sucursal bancaria).
- ✓ Fecha de inicio
- ✓ Fecha fin
- ✓ Acuerdo (número de consecutivos que representa el código del acuerdo en cuestión)

Prioridad: Alta

2.1.3. Especificación de requisitos no funcionales

Especifican criterios que pueden usarse para calificar las funcionalidades de un sistema en lugar de sus comportamientos específicos. Definen propiedades y restricciones del sistema. [20]

De acuerdo con las características sobre las cuales se desarrolla este subsistema se toman en cuenta los requisitos no funcionales (RNF) correspondientes al sistema Quarxo. Su descripción puede consultarse en el documento de tesis en formato digital (a partir de la *página 31*).

2.1.4. Especificación de casos de uso

Es un modelo del sistema que contiene actores, casos de uso y sus relaciones. Permite que los desarrolladores de software y el cliente lleguen a un acuerdo sobre las condiciones y posibilidades que debe cumplir el sistema. Describe lo que hace el sistema para cada tipo de usuario. Cada usuario se representa mediante uno o más actores. [33]

Mediante este modelo se pretende un acercamiento a los requisitos especificados visto desde un lenguaje más asequible a los desarrolladores.

Actores

Los actores representan terceros fuera del sistema que colaboran con el mismo. Al identificar los actores del sistema se identifica el entorno externo del sistema. Un actor juega un papel importante por cada CU con el que colabora. Cuando un usuario

(persona u otro sistema) interactúa con el sistema, el actor correspondiente está cometiendo ese papel.

En este caso el actor que representa al usuario o especialista bancario que interactúa con el sistema se llama usuario.

ACTOR DEL SISTEMA	DESCRIPCIÓN
Usuario	Generaliza a todos los especialistas dentro del BNC que interactúan con el subsistema Reportes.

Tabla 2. Descripción del actor

Diagrama de casos de uso del sistema

La vista de los casos de uso y con ella el diagrama de casos de uso modela cada funcionalidad del sistema según lo perciben los usuarios externos (actores). Un CU es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El designio de esta vista de casos de uso es enumerar a los actores y los casos de uso, y señalar qué actores participan en cada CU.

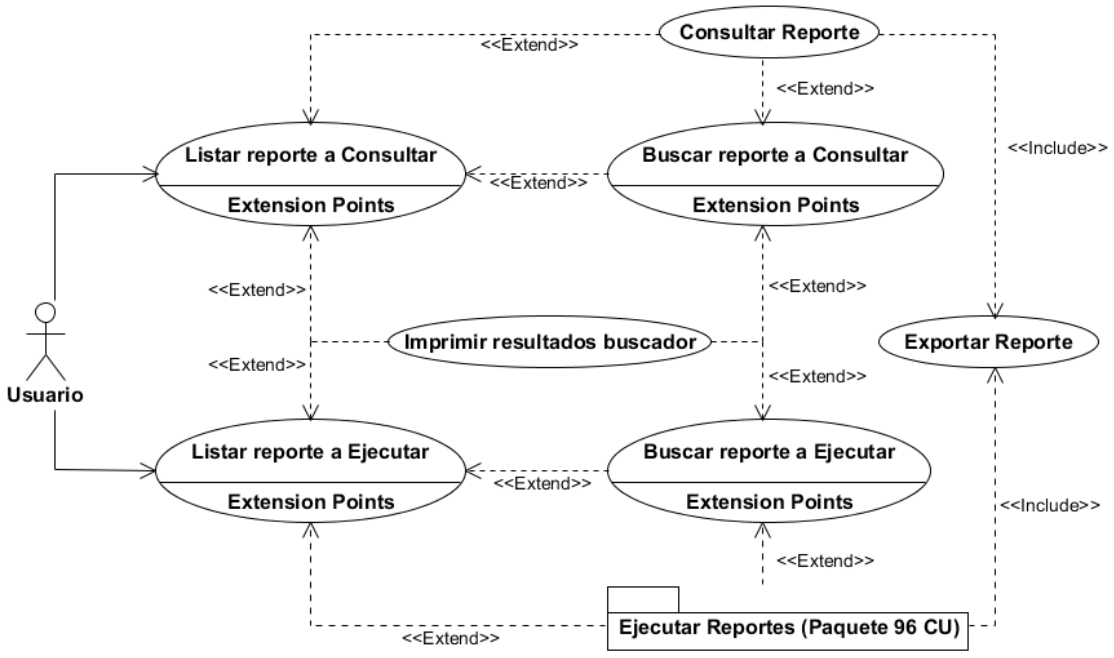


Figura 2. Diagrama de casos de uso del sistema.

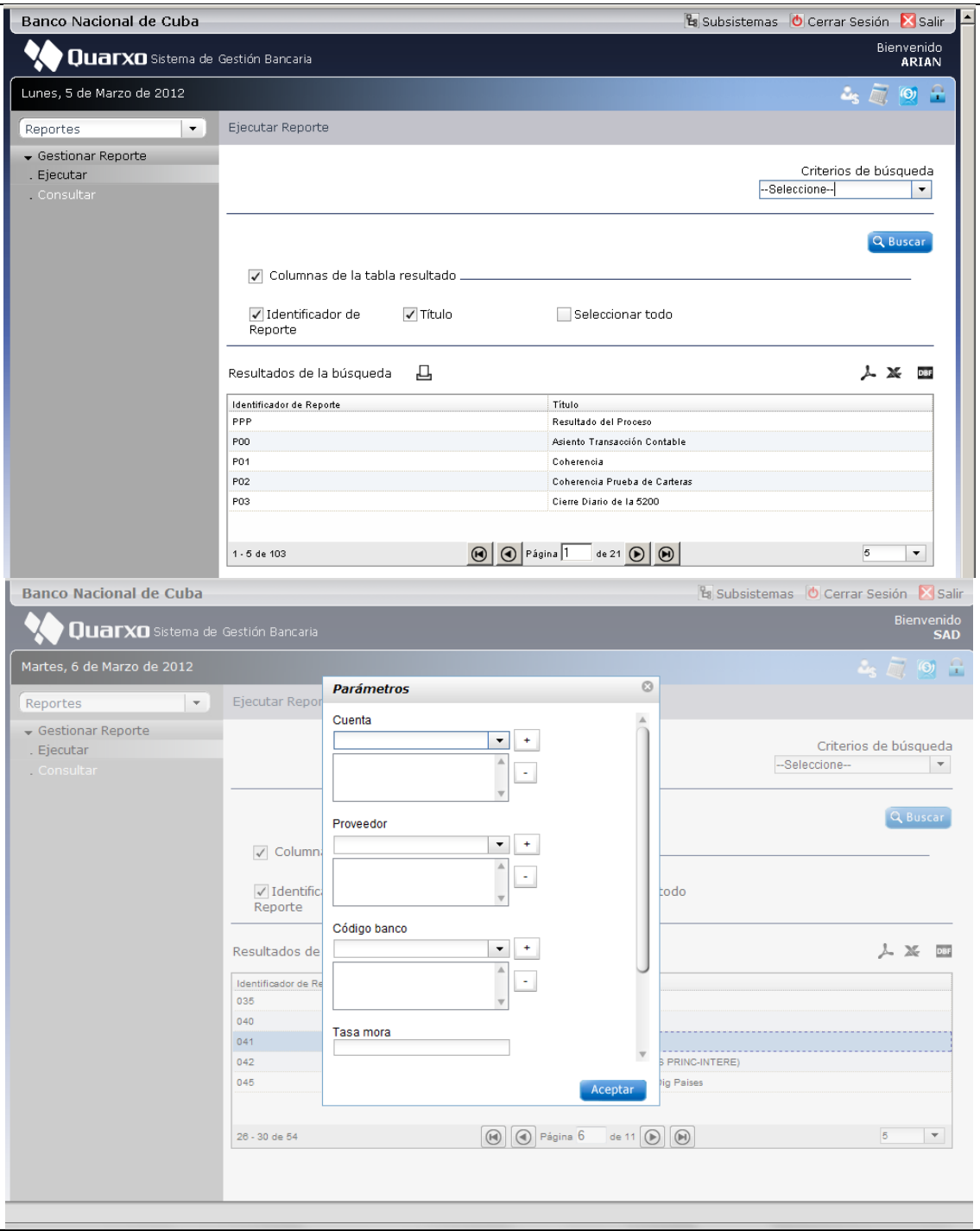
Descripción de los casos de uso del sistema

La forma en que los actores usan el sistema se representa mediante un CU. Se entiende por ellos como pedazos de funcionalidad que el sistema brinda para contribuir con un resultado de valor para sus actores. Un CU describe una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.

Los casos de uso se pueden describir en varios niveles de detalle. Se pueden agrupar partes así como igualar funcionalidades en aras de convertirlos en casos de uso más simples. Un CU se implementa como una colaboración en la vista de interacción.

A continuación se muestra la descripción de uno de los casos de uso más críticos del sistema, CU: Ejecutar reporte. Las restantes descripciones se pueden consultar en el documento de tesis en formato digital (*página 36 y Anexo # 2*).

CASO DE USO	Ejecutar Reporte
ACTORES	Usuario
RESUMEN	El CU inicia cuando el actor selecciona la opción Ejecutar Reporte, mediante la cual puede obtener el reporte deseado, proporcionándole un determinado número de parámetros, si los lleva, o ejecutándolo directamente.
PRECONDICIONES	El usuario debe estar autenticado con los permisos necesarios.
REFERENCIAS	**RF** 09, **RF** 10
PRIORIDAD	Alta
FLUJO NORMAL DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. El usuario selecciona la opción “Ejecutar Reporte”.	2. El sistema muestra la interfaz “Buscar Reportes”.
3. El usuario introduce los parámetros para realizar la búsqueda y selecciona la opción “Buscar”.	4. El sistema realiza una búsqueda de los reportes asociados al área al que pertenece el usuario y los coloca en la interfaz “Ejecutar Reporte”
5.El usuario selecciona el reporte a ejecutar	6. El sistema muestra el formulario para la ejecución del mismo.
7. El Usuario completa los campos y selecciona la opción “Aceptar”	8. El sistema valida los datos introducidos 9. El sistema persiste el lote asociado a dicho reporte 10. Se visualiza el reporte. Finaliza el CU.
PROTOTIPO DE INTERFAZ	



FLUJO ALTERNO AL PASO 4 “BÚSQUEDA FALLIDA“	
ACCIÓN DEL ACTOR	**RESPUESTA DEL SISTEMA**
	4.1. El sistema no encuentra ningún reporte en relación a los parámetros de búsqueda introducidos. Regresa al paso 3 del flujo normal de eventos.
FLUJO ALTERNO AL PASO 6 “OPERACIÓN CANCELADA”	
ACCIÓN DEL ACTOR	**RESPUESTA DEL SISTEMA**

6.1. El usuario selecciona la opción “Cancelar”.	6.2. El sistema cancela la operación y regresa al paso 4 del flujo normal de eventos.
FLUJO ALTERNO AL PASO 7 “DATOS INCORRECTOS Y/O CAMPOS VACÍOS”	
ACCIÓN DEL ACTOR	**RESPUESTA DEL SISTEMA**
	7.1. Durante la validación el sistema detecta datos incorrectos y/o campos obligatorios vacíos.
	7.2. El sistema muestra símbolos de error resaltando los campos obligatorios vacíos y/o donde se introdujeron datos incorrectos. Regresa al paso 5 del flujo normal de eventos.
PROTOTIPO DE INTERFAZ	
POSCONDICIONES	

Tabla 3. Descripción del CU Ejecutar reporte.

2.1.5. Validación de requisitos

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

- ✓ Validación de requisitos mediante prototipos y escenarios

Se presentaron los prototipos elaborados durante la especificación a especialistas funcionales, para corroborar que responden a las necesidades y aspiraciones del cliente. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaron las diferentes funcionalidades que tendría el subsistema. Las no conformidades fueron documentadas y corregidas.

- ✓ Revisión técnica de artefactos

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

2.2. El análisis

El análisis permite estudiar y comprender con mayor facilidad los requisitos que se describieron anteriormente en la captura de requisitos. Este estudio permite alcanzar una comprensión más exacta así como una descripción de los mismos que sea fácil de

interpretar y entender, apoyando de esta manera la estructuración del sistema completo. [33]

El propósito fundamental de esta etapa es resolver una serie de cuestiones que giran en torno a los casos de uso mediante el análisis de los requisitos con mayor profundidad. La mayor diferencia existente entre la captura de requisitos y el análisis está dada en que el primero se realiza utilizando el lenguaje del cliente, mientras que la segunda se basa fundamentalmente en el lenguaje de los desarrolladores. [33]

2.2.1. Modelo de análisis

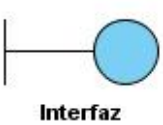
En un nivel técnico, la ingeniería del software empieza con una serie de tareas de modelado que llevan a una especificación completa de los requisitos y a una representación del diseño general del software a construir. El modelo de análisis, se representa mediante un sistema de análisis que denota el paquete de más alto nivel del modelo. Debe lograr tres objetivos primarios:

- ✓ Describir lo que requiere el cliente.
- ✓ Establecer una base para la creación de un diseño de software
- ✓ Definir un conjunto de requisitos que se pueda validar una vez que se construye el software. [34]

2.2.1.1. Clases del análisis

Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema. Esta abstracción posee las siguientes características:

- ✓ Se centra en el tratamiento de los requisitos funcionales.
- ✓ Las clases de análisis siempre encajan en uno de tres estereotipos básicos: de interfaz, de control o de entidad. [34]

NOMBRE	CARACTERÍSTICAS	ESTEREOTIPOS
Interfaz	Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores. Esta información a menudo implica recibir y presentar información y peticiones de y hacia los usuarios y los sistemas externos.	

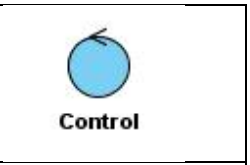
Control	Las clases de control representan coordinación, secuencia, transacciones, y control de otros objetos, y se usan con frecuencia para encapsular el control de un CU en concreto.	
Entidad	Las clases de entidad se utilizan para modelar información que posee una larga vida y que es a menudo persistente. Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto o un suceso del mundo real.	

Tabla 4. Clases del análisis.

2.2.1.2. Diagramas de clases del análisis

El diagrama de clases del análisis es un diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Estos diagramas de clases son utilizados durante el proceso de análisis y el diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargan del funcionamiento y la relación entre uno y otro. [34]

Los diagramas de clases del análisis y diagramas de colaboración del subsistema Reportes se pueden consultar en el documento de tesis en formato digital (a partir de la *página 40 y Anexo # 3*).

2.2.1.3. Descripción de la arquitectura

El sistema informático Quarxo se desarrollará bajo la tecnología JEE, este constituye el primer elemento distintivo de la arquitectura definida. Está compuesta por una arquitectura de tres capas: presentación, negocio y acceso a datos; además de una capa transversal a las otras con los objetos del dominio.

Capa de presentación

En esta capa se desarrollará la lógica de presentación. En el servidor se utiliza *Spring* MVC para recibir, controlar y enviar una respuesta a las peticiones realizadas desde el cliente. En el cliente se utiliza la librería *Dojo Toolkit* para generar las interfaces que interactuarán con el usuario. La capa de presentación estará relacionada con la Capa de Negocios y Capa de Dominio.

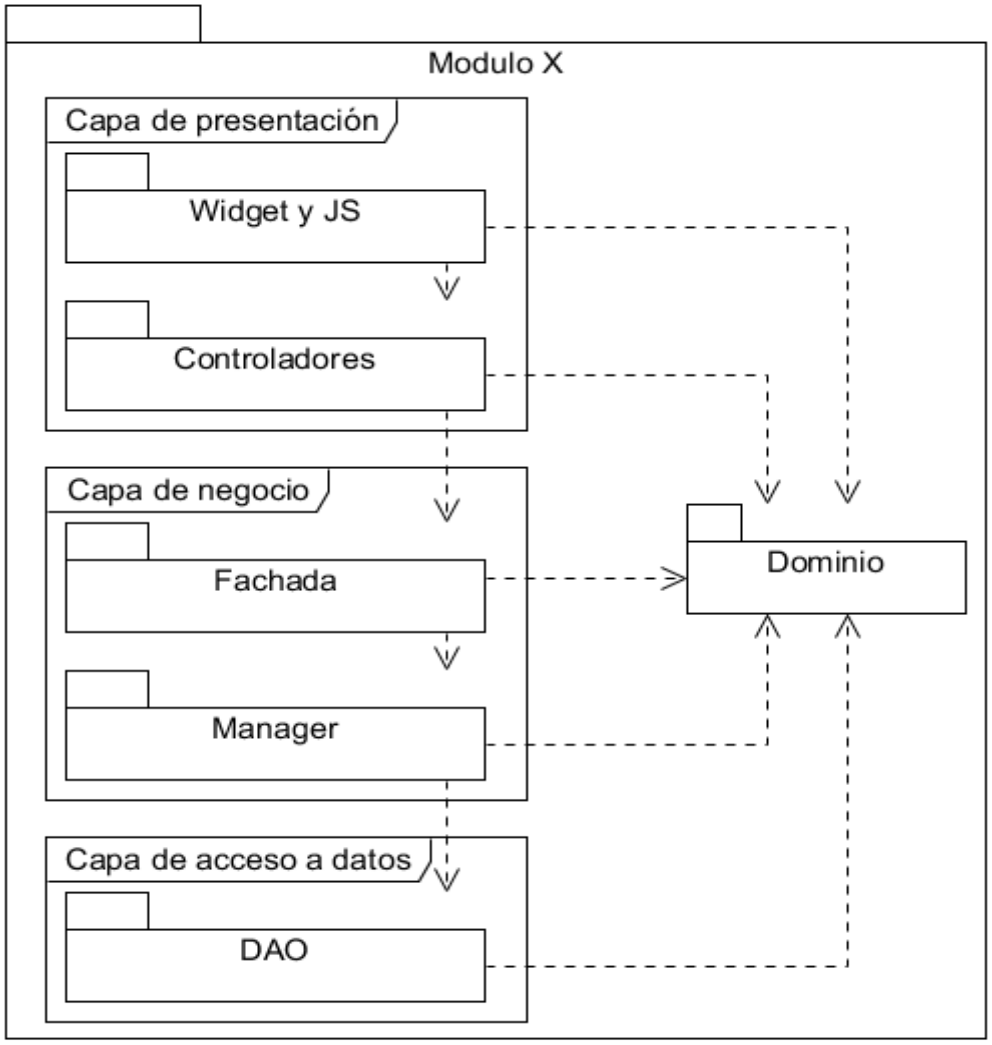


Figura 3. Arquitectura del Sistema.

Capa de negocio

Esta capa estará dividida en dos subcapas: Fachada y Manager. La Fachada será el punto de intercambiando entre la capa de presentación y la capa de negocio. Esta capa no tendrá lógica de negocio sino que agrupará funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación. La subcapa Fachada delegará a la subcapa Manager la realización de la lógica del negocio.

Por otro lado, la subcapa Manager tiene la jerarquía de clases suficiente para implementar el negocio de la aplicación. Esta subcapa utilizará la capa de Acceso a Datos para obtener los datos persistidos y la capa de Dominio para generar los objetos del dominio.

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

Capa de acceso a datos

En esta capa se encuentran las operaciones que permiten la interacción con el gestor de base de datos desde la aplicación. Desde aquí se ejerce la conexión con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información. La interacción con la capa de negocio se realiza a través de interfaces. Se utiliza el patrón DAO para el desarrollo de la capa, el *framework* de persistencia *Hibernate* y los módulos *Spring ORM* y *Spring DAO*. *Hibernate* como tecnología ORM para el manejo objetual de acciones de persistencia con la base de datos. *Spring ORM* y *Spring DAO* para soportar la integración con *Hibernate* y la utilización del patrón DAO.

La arquitectura definida presenta como otra de sus características la utilización del patrón MVC, patrón que propone dividir la aplicación en tres capas distintas, el Modelo, la Vista y el Controlador, potenciando la flexibilidad y la adaptabilidad a futuros cambios. Específicamente el Modelo es la representación de la información que maneja la aplicación, la Vista constituye la representación del modelo en forma gráfica disponible para la interacción con el usuario y el Controlador se encarga de responder a las solicitudes del usuario desde la interfaz, manejando los diferentes eventos a través de las funcionalidades necesarias y la información perteneciente al Modelo. [27]

2.2.1.4. Paquetes del análisis

Los paquetes del análisis propician un medio para organizar los artefactos del modelo de análisis en fragmentos más fáciles de manejar. Muestran una abstracción más concreta de lo que sería la composición de los paquetes de las diferentes clases y los casos de uso. [33]

En este diagrama se incluyen las plantillas de los reportes, las cuales contienen la estructura base para su generación. Los paquetes expuestos constituyen en gran

medida la estructura general del componente que gestiona los reportes y del subsistema que maneja sus funciones.

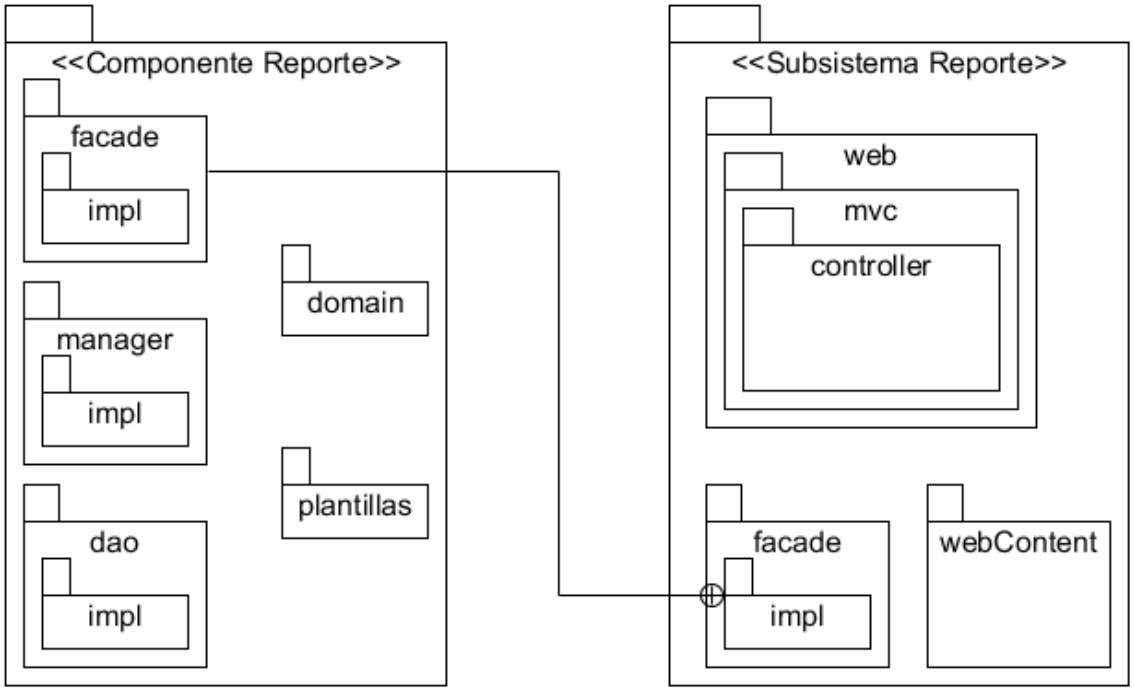


Figura 4. Distribución de los paquetes en el componente y el subsistema.

2.3. El diseño

En el diseño se determina la arquitectura general del sistema y su comportamiento dinámico, adaptando la especificación realizada en la etapa anterior. En esta fase se establece el comportamiento dinámico del sistema, es decir, cómo debe reaccionar ante los acontecimientos. [33]

El resultado obtenido de la etapa de Diseño facilita enormemente la implementación posterior del sistema, pues proporciona la estructura básica del sistema y cómo los diferentes componentes actúan y se relacionan entre sí.

2.3.1. Modelo de diseño

En el flujo de trabajo de Análisis y Diseño, definido por RUP, se describe cómo se debe implementar el sistema, como resultado de un análisis realizado a los requerimientos no funcionales del mismo. El diseño no debe tener ambigüedades para que el modelo final que se obtenga sea suficiente para la implementación. El modelo de diseño es el resultado más importante de este flujo de trabajo, el cual consiste en

colaboraciones de clases, las que pueden ser agrupadas en paquetes y subsistemas, además describe la realización de los casos de uso. [33]

2.3.1.1. Diagrama de clases del diseño

En el diseño se modela el sistema para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Se muestra el diagrama del diseño donde se describe la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. [33]

Este diagrama es muy utilizado durante el diseño de los sistemas, y a partir de él se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. A continuación se muestra el diagrama de clases del diseño del CU Ejecutar Reportes, los diagramas restantes se pueden consultar en el documento de tesis en formato digital (*en el Anexo # 4*).

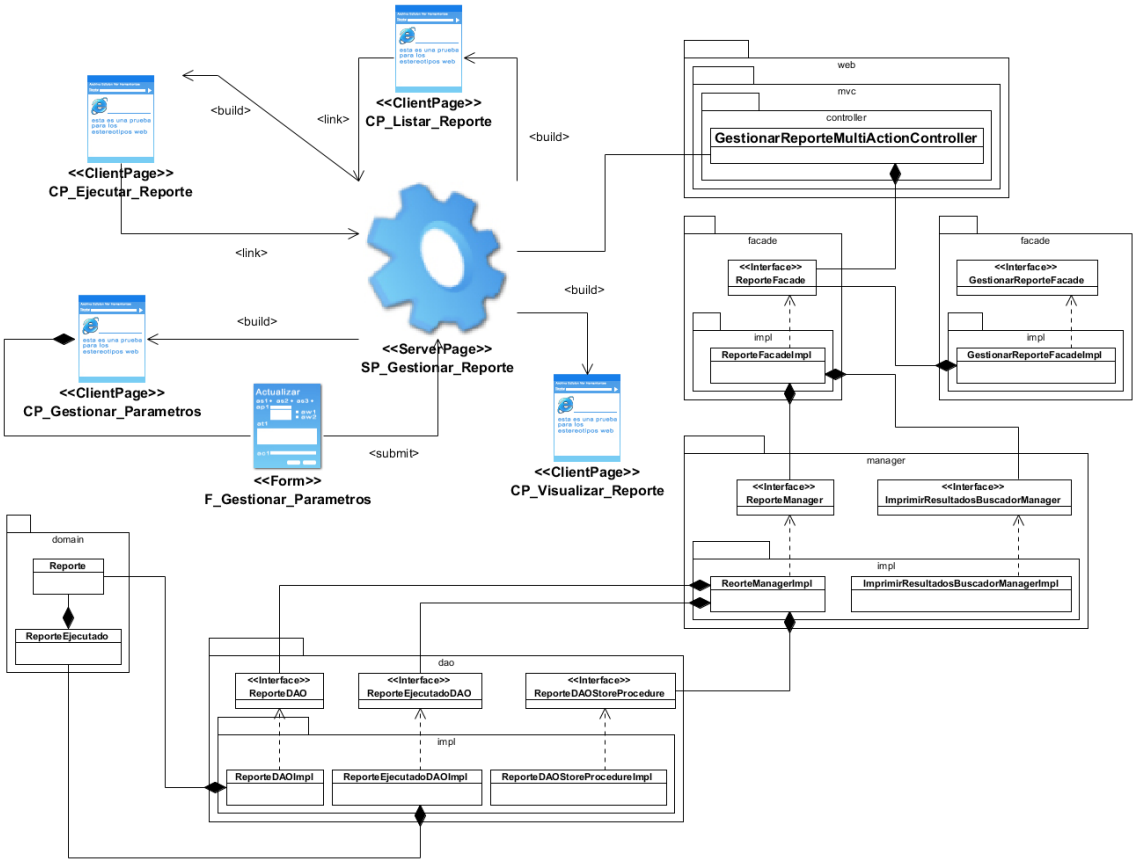


Figura 5. Diagrama de clases del diseño del CU Ejecutar Reporte.

Plantillas

Una vez creados los reportes, son almacenados en la aplicación dentro del paquete “plantillas”. Desde esta dirección son llamados para realizar las ejecuciones de los mismos. Este paquete se encuentra ubicado en el componente de reportes que está asociado al subsistema, de modo que se pueda acceder a él desde cualquier parte del sistema. Esto posibilita una independencia estructural del componente permitiendo que el mismo pueda ser utilizado para otras aplicaciones con los mismos fines.

Como ya se explicó anteriormente mediante la herramienta *IReport* se realiza el diseño de los reportes para posteriormente ser insertados en la aplicación. El proceso de elaboración de los reportes consta de tres pasos fundamentales:

- ✓ **Análisis del reporte:** fue realizado en la captura de requisitos con los especialistas del BNC.
- ✓ **Diseño del reporte:** está sujeto a la utilización del *IReport* para dotar al reporte con todos los requisitos analizados.
- ✓ **Implementación del reporte:** se realiza mediante una o varias consultas sobre las tablas de la Base de Datos utilizando el *SQLServer*. Dicha consulta se almacena en el diseño del reporte para que posteriormente sean almacenados los dos ficheros por los cuales está compuesto el reporte: estructura (*.jrxml) y el compilado (*.jasper).

El siguiente diagrama de actividades muestra el orden de los pasos explicados.

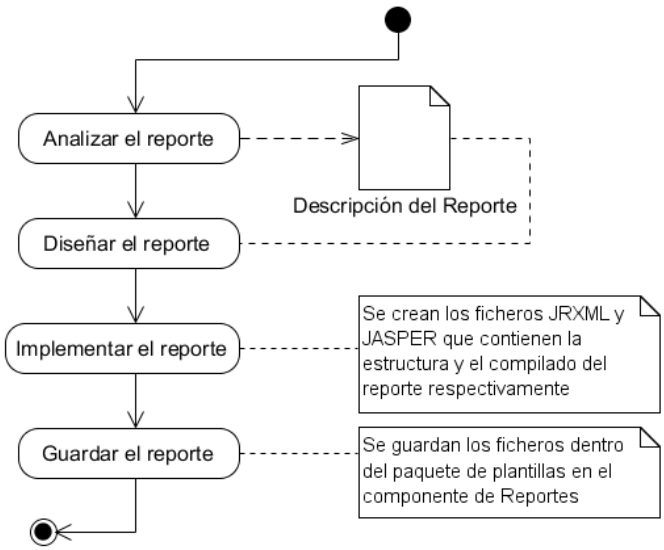


Figura 6. Secuencia de actividades para la creación de un reporte.

2.3.1.2. Diagrama de despliegue

Es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño. Este modelo representa una correspondencia entre la arquitectura del software y la arquitectura del sistema. [33]

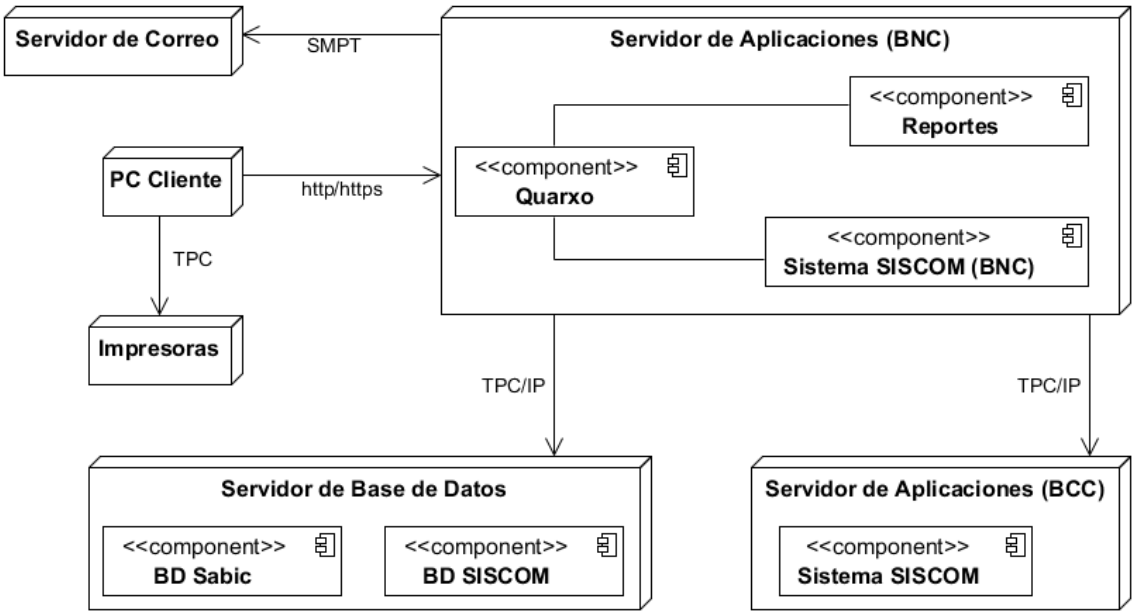


Figura 7. Diagrama de despliegue del sistema Quarxo.

2.3.2. Patrones

2.3.2.1. Patrones de arquitectura

Dividir un sistema en capas tiene una cantidad importante de beneficios y tanto en el componente como en el subsistema se evidencian dichos beneficios:

- ✓ Se puede entender una capa como un conjunto, sin considerar las otras.
- ✓ Las capas se pueden sustituir con implementaciones alternativas de los mismos servicios básicos.
- ✓ Se minimizan dependencias entre capas.
- ✓ Las capas posibilitan la estandarización de servicios.
- ✓ Luego de tener una capa construida, puede ser utilizada por muchos servicios de mayor nivel.

2.3.2.2. Patrones de diseño

El desarrollo de un sistema conlleva, en la mayoría de las ocasiones, a darle solución a problemas muy complejos que ya alguien ha resuelto con anterioridad. Por esta razón uno de los pasos a tener en cuenta cuando se decide desarrollar un proyecto de software es identificar qué patrones pueden ser utilizados. Entiéndase por patrón como una solución estándar para un problema común de programación. A continuación se muestran los patrones aplicados en el subsistema Reportes.

Durante el diseño del componente se emplearon patrones GRASP, específicamente:

- ✓ **Controlador:** La clase controladora *GestionarReporteMultiActionController*, constituye un ejemplo de la aplicación de este patrón, la misma tendrá en cuenta la responsabilidad de manejar los eventos que consisten en gestionar los reportes en el subsistema.
- ✓ **Experto:** Se evidencia en la definición de las clases de acuerdo con las funcionalidades que deben realizar a partir de la información que manejan. Específicamente: la clase *ReporteDAO* y *ReporteEjecutadoDAO*, serán las responsables de efectuar las operaciones que corresponden a la gestión de los reportes.
- ✓ **Bajo acoplamiento:** Se evidencia con la definición de interfaces e implementaciones, como la interfaz *ReporteFacade* y su implementación, que permiten que *GestionarReporteMultiActionController* interactúe con las clases de presentación y se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

Durante el diseño del componente se emplearon patrones GoF, específicamente:

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

Patrón de Acceso a Datos (*Data Access Object*)

- ✓ **DAO:** Se evidencia en la definición de las clases interfaces *ReporteDAO*, *ReporteEjecutadoDAO* y *ReporteDAOStoreProcedure* cada una con sus respectivas implementaciones, en las cuales se facilitan las funcionalidades específicas a realizar sobre la base de datos. De esta manera, el negocio no será afectado de posibles cambios que puedan producirse en la lógica de acceso a datos y la fuente de datos.

2.4. Conclusiones del capítulo

- ✓ El proceso de desarrollo de software fue guiado por la metodología RUP, cumpliendo con los elementos que dicha metodología propone y logrando una efectiva ingeniería de software.
- ✓ Se desarrollaron los flujos de trabajo Requisitos, Análisis y Diseño propuestos por la metodología, obteniendo el Modelo de casos de uso, Modelo de análisis y Modelo de diseño, como principales artefactos en cada caso.
- ✓ Se describió la arquitectura basada en el MVC, fundamentada en la utilización de los *frameworks Spring*, *Hibernate* y *Dojo Toolkit*, y los patrones considerados para el diseño de los casos de uso.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SUBSISTEMA REPORTES DE QUARXO

Introducción

En el presente capítulo se analizan los flujos de trabajo implementación y pruebas, correspondientes a la metodología RUP destacándose la modelación de los artefactos más importantes respectivamente. Se expone la nomenclatura usada tanto en el código como en los paquetes y clases del subsistema, y se abordan algunos temas importantes sobre la implementación como la utilización de librerías para la gestión de los reportes. Se muestran fragmentos de código de los principales flujos de la implementación, así como los métodos más relevantes.

3.1. Implementación

La implementación es el principal flujo de trabajo en la fase de construcción. En él se describe cómo los elementos del modelo de diseño se implementan en función de componentes y por ende en piezas más manejables por el lenguaje del programador. Tiene como objetivo llevar a cabo la implementación de cada una de las clases significativas del diseño. [34]

3.1.1. Modelo de implementación

Este modelo detalla la implementación de los elementos del modelo de diseño (clases), en términos de componentes (ficheros de código fuente y ejecutables). Expone la organización de estos de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación usado y la dependencia entre los propios componentes. [33]

3.1.1.1. Diagrama de componentes

Muestran un conjunto de componentes y sus relaciones. Se corresponde con una o más clases, interfaces o colaboraciones. El siguiente diagrama muestra los componentes del subsistema siguiendo el patrón de arquitectura Modelo Vista Controlador y su interacción con el resto de componentes dentro del sistema.

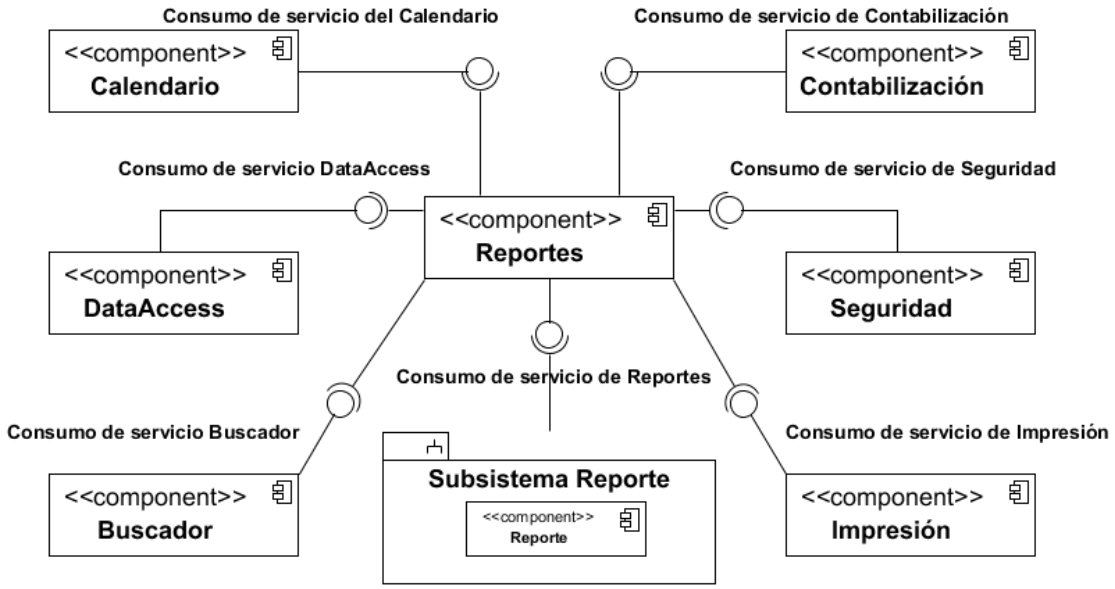


Figura 8. Diagrama de componentes asociado al subsistema Reportes.

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

El componente **Calendario** sirve para precisar el tiempo, los períodos y los montos en los que serán emitidos algunos de los reportes del BNC, los cuales pueden estar implícitos en la emisión de un reporte.

El componente **Buscador** ofrece funcionalidades para la correcta gestión de la información en el subsistema Reportes, se hace necesaria la búsqueda tanto de reportes como de lotes (reportes ejecutados).

El componente **Contabilización** permite manipular algunos reportes como procesos contables. El proceso de contabilización requiere de un reporte que se genera a partir de los datos producidos por los procesos contables.

El componente **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 la realice.

El componente **Impresión** tiene como función brindar funcionalidades mediante las cuales es posible imprimir determinada información de forma automática. El subsistema Reportes tiene entre sus funcionalidades la impresión de los reportes.

El componente **DataAccess** tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos.

3.1.1.2. Descripción de la arquitectura

Descripción de clases y funcionalidades

A continuación se describen las funcionalidades de la clase más significativa desde el punto de vista del negocio para el subsistema Reportes. Las restantes descripciones de clases se pueden consultar en el documento de tesis en formato digital (*página 55*).

NOMBRE	ReporteManagerImpl
TIPO DE CLASE	Manager
ATRIBUTOS	**TIPO DE DATO**
dataSource	DataSource
reporteDAO	ReporteDAO
reporteEjecutadoDAO	ReporteEjecutadoDAO
reporteDAOStoreProcedure	ReporteDAOStoreProcedure
PARA CADA RESPONSABILIDAD	
NOMBRE	**DESCRIPCIÓN**
listadoReportes(int inicio, int cantidad)	Devuelve una lista de reportes y dados esos parámetros, se utiliza para llenar la tabla paginada de la vista.
listarLotes()	Devuelve una lista con los lotes almacenados en la base de datos.
emitirReporte(String identificador, Map parameters, String usuario, String formato)	Se encarga de ejecutar un reporte dado el identificador, los parámetros, el formato deseado, y el código del usuario que lo solicita respectivamente.
obtenerReporte(String título)	Permite obtener un reporte dado el título del mismo.
consultarLote(String idLote)	Permite consultar un lote dado su identificador.
gestionarParametros(String id)	Permite gestionar la estructura y configuración de los parámetros correspondientes a un reporte dado su identificador.
tablaDatos(String tabla, String cod, String valor)	Permite cargar los datos multivaluados de la base de datos para los parámetros que lo requieran.
obtenerReporteByID(String id)	Permite obtener un reporte dado el identificador del mismo.

obtenerLote(String id)	Permite obtener un lote dado el identificador del mismo.
cantidadReportes()	Permite obtener la cantidad de reportes disponibles a ejecutar.
listarPaginado(String tabla, String campoBuscar, String campoMostrar, int inicio, int cantidad, String value)	Permite cargar los reportes a la tabla de ejecución de forma paginada.
exportarDBF(String id)	Permite conocer si un reporte seleccionado puede ser exportado en formato DBF.
reservarConsecutivo()	Permite reservar un consecutivo de la tabla M_CONSEC para cada reporte que se ejecute.
persistirContabilizacion(ReporteEjecutado lote)	Permite guardar en la base de datos el reporte correspondiente a una contabilización realizada
construirReporte(String consec, String identificador, Map parameters, Resource r[], Connection cxn)	Dado sus parámetros permite localizar el fichero del reporte y lo ejecuta dando como resultado un arreglo de *bytes* con el lote correspondiente.
guardarLote(Reporte report, Blob b, String consec, String format, Map parameters, Resource r[], Connection cxn, Session s)	Permite guardar el lote una vez que fue emitido completamente el reporte.
obtenerEstructuraReporte(JasperPrint print)	Se encarga de crear una estructura basada en las reglas del formato DBF para exportar un reporte con esta extensión (*.dbf)
crearReporteDBF(List<Map> fila, List<List<String>> data, String title)	Una vez construida la estructura para el DBF este método se encarga de llenar el DBF con los datos capturados en la emisión del reporte.

Tabla 5. Descripción de la clase ReporteManagerImpl.

3.1.2. Estándares de codificación

3.1.2.1. Convenciones de nomenclatura

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto con minúscula, en caso de ser una palabra compuesta se empleará la notación *PascalCasing*[2].

Ejemplo: *ReporteManager.java, ReporteFacadeImpl.java*

[2] Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula.

Los nombres de los métodos y los atributos de las clases, así como los nombres de ficheros de código *JavaScript* y sus funciones y variables internas comienzan con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing*[3].

Ejemplo: *emitirReporte, dataSource.*

Los nombres de las plantillas de los reportes que se almacenan en el componente se han organizado de la siguiente forma: El nombre de los ficheros que componen el reporte (*.jrxml, *.jasper) tiene un prefijo "*RPT_*" seguido por una numeración de tres dígitos o tres letras del alfabeto en mayúscula correspondiente al identificador en la tabla *o_reporte.*

Ejemplo: *RPT_040.jrxml, RPT_040.jasper, RPT_IMC.jrxml, RPT_IMC.jasper.*

3.1.2.2. Nomenclatura según el tipo de clase

Las clases que se encuentran dentro del paquete *controller* se nombran adicionándoles el nombre del controlador de *Spring,* del cual heredan al final del nombre de la clase (*MultiActionController*).

Ejemplo: *GestionarReporteMultiActionController.*

Las clases que se encuentran dentro del paquete *facade* se nombran adicionándoles como sufijo del nombre la palabra *Facade*. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*.

Ejemplo: *ReporteFacade, ReporteFacadeImpl.*

Las clases que se encuentran dentro del paquete *manager* se nombran adicionándoles como sufijo del nombre la palabra *Manager*. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*.

Ejemplo: *ReporteManager, ReporteManagerImpl.*

[3] Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula excepto la primera palabra.

Las clases que se encuentran dentro del paquete *dao* se nombran adicionándoles como sufijo del nombre las siglas DAO o *DAOStoreProcedure* dependiendo de la fuente de datos origen. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*.

Ejemplo: *ReporteDAO, ReporteDAOImpl, ReporteDAOStoreProcedure, ReporteDAOStoreProcedureImpl.*

Las clases que se encuentran dentro del paquete *report* tienen dos nomenclaturas: una mediante la utilización de la palabra *Imprimir* como prefijo del nombre de la clase y la otra mediante la palabra *DataSource* como sufijo del nombre de la clase. De esta forma tenemos por cada operación una pareja de clases relacionadas entre ellas.

Ejemplo: *ImprimirAsientos.java, AsientosDataSource.java.*

Dentro del paquete *resource* que se encuentra en el paquete *report* se almacenan los reportes asociados a las clases del paquete *report.* Su nomenclatura es similar a la de los otros reportes, después del prefijo “RTP_” se agrega el nombre completo del reporte.

Ejemplo: *RPT_Asiento Transaccion Contable.jasper, RPT_Asiento Transaccion Contable.jrxml.*

3.1.3. Aspectos fundamentales de la implementación

En esta sección se analizan algunas de las librerías más importantes usadas en el proceso de implementación del subsistema Reportes de Quarxo.

3.1.3.1. Utilización de la librería JasperReports para generar y exportar los reportes

La piedra angular del desarrollo del subsistema Reportes se basa en la manera en que se gestionan los reportes para la aplicación. Esto se hace posible mediante la utilización de la librería *JasperReports* que permite obtener y generar los reportes de manera rápida y organizada. A continuación se describen algunos de los segmentos de código, pertenecientes a varios de los métodos encargados de la gestión de los reportes.

La principal función del componente de reportes es la emisión de los reportes del BNC. Este proceso se logra mediante la creación de un arreglo de *bytes* que recoge los datos del compilado del reporte (fichero *.jasper) que se encuentra en el paquete *plantillas*. El arreglo se llena a partir del método *runReportToPdf* de la clase *JasperManajer,* el cual llena un informe y lo guarda directamente a un fichero PDF (ver figura 9).

Dicho método recibe una URL que contiene la dirección del compilado, un objeto *Map* con los parámetros con los cuales se va a ejecutar el reporte y por último un objeto *Connection* para establecer la conexión entre el reporte y la fuente de datos. El arreglo de *bytes está contenido* en una estructura de un PDF que contiene la emisión del reporte ya que es un formato común y cómodo de usar y manejar.

```
byte array[] = { 0 };
for (Resource resource : r) {
    array = JasperRunManager.runReportToPdf(resource.getURL().getPath()
            .replace("%20", " "), parametros, cxn);
}
return array;
```

Figura 9. Segmento de código del método *emitirReporte*. Acceso al reporte.

Una vez que se tiene el arreglo de *bytes* se guarda en la base de datos como un lote y se le presenta al usuario en el formato solicitado: PDF, EXCEL o DBF.

Cuando la previa selección del formato ha sido DBF o EXCEL los datos del reporte deben ser leídos de otra manera. Para esto se utiliza el método *fillReport* de la clase *JasperFillManager* que recibe los mismos parámetros que el *runReportToPdf,* pero que llena el diseño del informe elaborado cargado desde el archivo especificado y devuelve el objeto de informe generado en un *JasperPrint* (ver figura 10).

```
if (formato.equals("pdf") || formato.equals("ver")) {
    return lote;
} else {
    if (formato.equals("dbf") || formato.equals("excel")) {
        JasperPrint print = null;
        for (Resource resource : r) {
            print = JasperFillManager.fillReport(resource.getURL()
                    .getPath().replace("%20", " "), parametros, cxn);
        }
```

Figura 10. Segmento de código del método *emitirReporte*. Verificación de los formatos a exportar.

En caso de exportar a *EXCEL* se emplea un objeto de la clase *JRExporter* instanciado como un *JRXlsExporter* que permite exportar un documento *JasperReports* a formato XLS. Tiene salida binaria y exporta el documento a un diseño basado en una tabla para obtener los parámetros propios del formato deseado y mediante el método *setParameter,* de la propia clase, se ajustan los detalles deseados para los documentos a crear, tales como: la cantidad de información por hoja de cálculo, el tipo de fondo y el tipo de letra.

3.1.3.2. Utilización de la librería DynamicJasper para imprimir los resultados del buscador

Es una extensión que permite generar reportes en forma totalmente programática, evitando tocar archivos *.jrxml. Para lograr esto se utiliza una instancia de la clase *FastReportBuilder* usada frecuentemente en la creación de reportes dinámicos (ver figura 11). Sobre esta instancia se crean las páginas, marcos y demás estilos necesarios para darle formato al reporte que se confecciona.

```
FastReportBuilder drb = new FastReportBuilder();
drb.setPageSizeAndOrientation(Page.Page_Letter_Portrait());
drb.setTemplateFile(templatePath);
int width = 95;
```

Figura 11. Creación de la estructura del reporte mediante DynamicJasper.

La instancia creada de *FastReportBuilder* permite además insertar dinámicamente tantas columnas como sea necesario. El caso que ocupa la utilización de esta librería se basa en la creación de un reporte a partir de los datos arrojados por el buscador del subsistema. La complejidad de esta operación consiste en que la cantidad de columnas a colocar en el reporte dependen de lo que estime conveniente seleccionar el usuario, por tanto, se precisa la utilización de DJ para que resuelva este problema generando columnas dinámicamente en el reporte. Por último, se guarda la estructura del reporte así como los datos insertados dentro de los campos configurados. Al final se obtiene un reporte en formato PDF que contiene los datos referentes a una búsqueda realizada.

3.1.3.3. Utilización de la librería JavaDBF para construir los reportes en formato DBF

Los archivos con formato DBF son ampliamente utilizados en muchas aplicaciones que necesitan un formato simple para almacenar datos estructurados. Representan tablas o estructuras en forma de tabla que requieren organización. Un DBF está compuesto por campos, los cuales representan los encabezados de cada columna en la tabla, y los datos.

Para crear una estructura DBF es necesario definir qué elementos contendrá cada campo así como la estructura del campo en sí. Se crea un objeto *DBFField* que contendrá un arreglo de campos con la información correspondiente a cada columna del DBF (ver figura 13), tales como: nombre del campo, tipo de dato del campo, longitud del campo y cantidad de decimales (para números de punto flotante).

```
DBFField fields[] = new DBFField[fila.size()];

for (int i = 0; i < fila.size(); i++) {
    DBFField field = new DBFField();
    field.setName(fila.get(i).get("campo").toString());
    String temp = fila.get(i).get("tipo").toString();
    if (temp.equals("C")) {
        field.setDataType(DBFField.FIELD_TYPE_C);
        field.setFieldLength(Integer.parseInt(fila.get(i).get("long")
                .toString()));
    } else {
        if (temp.equals("D")) {
            field.setDataType(DBFField.FIELD_TYPE_D);
        } else {
            if (temp.equals("I") || temp.equals("N")) {
                field.setDataType(DBFField.FIELD_TYPE_F);
                field.setFieldLength(Integer.parseInt(fila.get(i).get(
                        "long").toString()));
                field.setDecimalCount(Integer.parseInt(fila.get(i).get(
                        "dec").toString()));
            } else {
                if (temp.equals("java.lang.Boolean")) {
                    field.setDataType(DBFField.FIELD_TYPE_L);
```

Figura 12. Creación de los campos del objeto DBF.

Una vez que se ha creado la estructura para los campos del DBF se usa un objeto de la clase *DBFWriter* para colocar los campos anteriormente llenados.

Posteriormente se agregan al *writer* los datos por cada campo verificando en cada caso que el dato corresponda con las características del campo donde va a ser colocado. Además, se crea un objeto de tipo *File* con los permisos pertinentes para luego confeccionar el fichero final con la extensión DBF.

3.2. Pruebas

El principal objetivo de esta disciplina es de evaluar la calidad del producto que se desarrolló. Se realiza a través de diferentes fases mediante la aplicación de pruebas concretas para validar que las suposiciones hechas en el diseño y que los requerimientos se estén cumpliendo satisfactoriamente.[33]

Esto significa que se verifica el funcionamiento del producto como se diseñó y que los requerimientos han sido cumplidos satisfactoriamente.

3.2.1. Pruebas unitarias

El objetivo de las pruebas unitarias es el aislamiento de partes del código y la demostración de que estas partes no contienen errores. Las pruebas unitarias son una forma de probar el correcto funcionamiento de un módulo de código, esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. [35]

Las pruebas unitarias realizadas a los servicios del sistema sirven para validar que las salidas son correctas y aseguran al desarrollador que la solución no presenta errores en la lógica de programación y que las respuestas son las correctas ante una entrada de datos determinada.

Se aplicarán los métodos de pruebas adaptados a este nivel, como son los métodos de prueba: Caja Blanca, para comprobar los caminos lógicos del software y Caja Negra, para probar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

3.2.1.1. Prueba de Caja Blanca

Para las pruebas de Caja Blanca se utilizó el *framework* JUnit, el cual brinda un conjunto de librerías que se integran fácilmente al IDE de desarrollo seleccionado: Eclipse. JUnit permite la realización de las pruebas a los métodos de las clases implementadas. Para hacer uso de este *framework* se definieron los siguientes pasos:

- ✓ Crear un caso de prueba por cada clase implementada.
- ✓ Configurar en el caso de prueba, los ficheros que permiten comunicar las clases de las capas de presentación y lógica de negocio.
- ✓ Definir los métodos a probar dentro de cada caso de prueba, incluyendo los parámetros de entrada.
- ✓ Realizar pruebas a los métodos. [36]

A continuación se muestran las imágenes que corresponden a la realización de las pruebas de Caja Blanca aplicando el *framework JUnit* al método más complejo de la implementación: *emitirReporte,* el cual se localiza en la clase controladora *GestionarReporteMultiActionController* dentro del subsistema Reportes.

Primeramente se crea un objeto de la clase donde se encuentra el método a probar y a continuación se da paso a crear tres *mocks*, debido a que el método recibe como parámetros dos objetos de tipo *HttpServlet(request y response)*. Esto lo posibilita la librería *EasyMock* que trabaja en la creación de un *proxy* dinámico para dicho simulacro, el cual es controlado a través de un objeto de tipo *MockControl* (ver figura 14)

```
public class GestionarReporteMultiActionControllerTest extends TestCase
{

    private GestionarReporteMultiActionController reporteMultiAction;

    private MockControl controlHttpServletRequest;
    private HttpServletRequest mockHttpServletRequest;

    private MockControl controlHttpServletResponse;
    private HttpServletResponse mockHttpServletResponse;

    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;
```

Figura 13. Declaración de las variables para la prueba.

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

Se inicializa cada *mock* creado para simular los objetos del método en cuestión. En este caso son tres *mock*: *mockHttpServletRequest, mockHttpServletResponse y mockHttpSession*.

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

Figura 14. Verificación de cada prueba.

El método *tearDown* (ver figura 14) es el encargado de las tareas a realizar en cada test. En este caso solo va a verificar los objetos que son pasados por el *request*.

Durante la realización de la prueba al método, se preparan los parámetros de pruebas necesarios al método *emitirReporte*. Dichos parámetros serán pasados por el *mockHttpServletRequest* (ver figura 15). Posteriormente se realiza la condición de prueba a través del método *assertTrue* el cual devuelve verdadero si se emitió el reporte de manera satisfactoria.

```
public void testEmitirReporte()
{
mockHttpServletRequest.getParameter("id");
controlHttpServletRequest.setReturnValue("008");

mockHttpServletRequest.getParameter("formato");
controlHttpServletRequest.setReturnValue("pdf");

mockHttpServletRequest.getAttribute("parametros");

Map map = new HashMap<String, String>();
List<Map> listMap = (List<Map>) new LinkedList<Map>();

controlHttpServletRequest.setReturnValue(listMap);
controlHttpServletRequest.replay();

assertTrue(reporteMultiAction.emitirReporte(mockHttpServletRequest, mockHttpServletResponse));

}
```

Figura 15. Parámetros utilizados por el método a probar.

El resultado de la prueba realizada a todos los métodos de la clase *GestionarReporteMultiActionController* fue satisfactorio (ver figura 16).

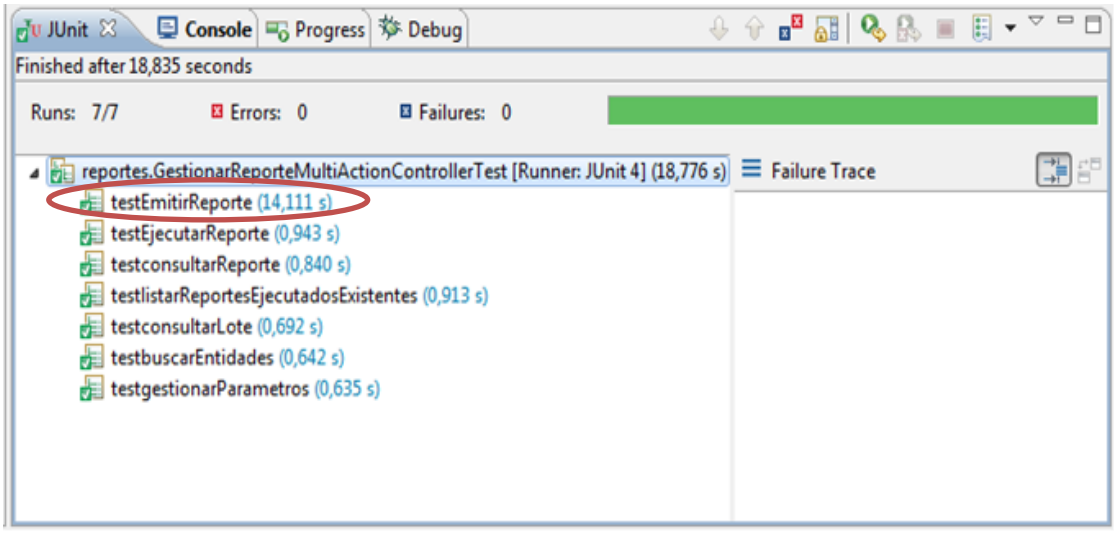


Figura 16. Resultado de la prueba realizada.

3.2.2. Prueba de Caja Negra

Las pruebas de Caja Negra o pruebas Funcionales se realizan sobre la interfaz del software, entendiendo por interfaz las entradas y salidas del mismo. No es necesario conocer su lógica de funcionamiento, únicamente la funcionalidad que debe realizar. También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una "Caja Negra" cuyo comportamiento solo puede ser determinado mediante el estudio de las entradas y salidas obtenidas a partir de ellas. [37]

No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable, se seleccionó un conjunto de ellas sobre las que se realizan las pruebas. Para seleccionar el conjunto de entradas y salidas se consideró que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en el sistema, y como consecuencia producen una o varias salidas que revelan la presencia de defectos o errores.

Las técnicas de prueba no se hallan de forma aislada, sino como un conjunto integrado de acciones que combinadas permitirán verificar y evaluar la calidad de software. Entre las técnicas para desarrollar la prueba de Caja Negra se encuentran:

- ✓ Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- ✓ Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- ✓ Técnica de Grafos de Causa-Efecto: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. [37]

De las técnicas mencionadas se aplicará la de **Partición de Equivalencia** ya que es una de las más efectivas. Permite definir casos de prueba que declaren clases de errores, reduciendo el número de casos de prueba a desarrollar para demostrar que las funciones del software son operativas; las entradas se aceptan de forma adecuada y se producen salidas correctas.

Para verificar que la aplicación se comporta según los requerimientos establecidos por el cliente, se diseñan casos de pruebas usando el método de Caja Negra. Para ello se realizan casos de pruebas para el CU "Ejecutar Reporte Pagos Efectuados

Contabilidad", los cuales se pueden consultar en el documento de tesis en formato digital (*Anexo # 6*).

Las pruebas realizadas al subsistema fueron satisfactorias desde el punto de vista interno y funcional, estas abarcaron requerimientos, funciones y lógica interna de cada módulo. Se aplicaron los métodos de Caja Negra y Caja Blanca para validar tanto la interfaz como el correcto funcionamiento interno del software.

3.2.3. Validación de las variables de la investigación

La investigación desarrollada plantea como idea a defender que: "Si se desarrolla el subsistema Reportes de Quarxo, se contribuirá a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución". A continuación se evalúan las variables tiempo de respuesta y recursos materiales con el uso del subsistema Reportes de Quarxo en el BNC.

3.2.3.1. Disminución de los tiempos de respuesta

El tiempo de respuesta se utiliza haciendo referencia a la demora en la resolución de una solicitud para crear o modificar un reporte. La disminución de esta variable implicaría necesariamente mayor eficiencia y productividad, a partir de que se dedicaría menos tiempo en solucionar estas operaciones, y se agilizarían los procesos que esperan por su realización.

Se consideraron las actividades que deben llevarse a cabo para responder a cualquiera que sea la solicitud, utilizando ambos sistemas. Esta comparación se resume en la tabla posterior.

ACTIVIDADES PARA LA CREACIÓN/MODIFICACIÓN DE REPORTES			
No	**ACTIVIDADES**	**SABIC**	**Subsistema Reportes**
1	Realizar consulta	✓ **Sin parámetros dinámicos:** restringe el reporte a un solo juego de datos, lo que implica la modificación del mismo para otros valores. ✓ **Sin agrupamientos:** la consulta debe diseñarse con los agrupamientos deseados, lo que implica mayor complejidad de la misma.	✓ **Parámetros dinámicos:** Permite colocar distintos juegos de datos en cada ejecución del reporte. ✓ **Agrupamientos:** No son necesarios en la consulta ya que el *IReport* permite hacerlos.
	OBSERVACIONES	Las consultas por lo general son extensas por lo que	

		realizar agrupamientos y cruzamientos hace que aumente su complejidad y tarde más su terminación.	
2	Crear el fichero con la consulta	✓ **Guardar consulta:** Está separada del diseño del reporte, se guarda en una dirección física del servidor. ✓ **Implementar fichero FRX[4]:**	No es necesaria esta actividad
	OBSERVACIONES	Las consultas están separadas del diseño del reporte por lo que un cambio en el diseño provoca la realización de esta actividad nuevamente para la implementación del fichero.	
3	Diseñar el reporte	✓ **Sobre MS-DOS:** Interfaz poco amigable para el diseño.	✓ ***IReport*:** Su uso posibilita la creación de reportes complejos de forma ordenada, rápida y eficiente.
	OBSERVACIONES	En SABIC, los campos deben ser colocados y alineados a simple vista o introduciendo valores de coordenada (x; y), mientras que *IReport* permite patrones de alineación y crea automáticamente los nombres de etiqueta de los mismos.	
4	Enlazar los campos con la fuente de datos	✓ **Diseño-Consulta:** Cada campo del diseño debe ser enlazado con el valor correspondiente en la consulta.	No es necesaria esta actividad
	OBSERVACIONES	Una vez que se tiene el diseño del reporte se necesita enlazar cada uno de los campos con el valor correspondiente en la consulta. Para realizar alguna modificación en esta actividad es necesario volver a la actividad dos.	

Tabla 6. Actividades para la creación y modificación de un reporte.

Escenario 1: Se realiza una prueba mediante la creación del reporte: "Vencimiento de Cartas de Créditos y Riesgos" por los dos sistemas.

1. Número de líneas de código en la consulta. (***IReport* – 192, SABIC - 247**)
2. Cantidad de agrupaciones en *IReport* (**5 agrupaciones** (País, Acuerdo, Mes, Año y Proveedor))
    - ✓ Esto provoca que la consulta en SABIC sea más larga y compleja.
3. Cantidad de parámetros (**9 parámetros** (País, Cliente, Acuerdo, Por Cuenta, Proveedor, Respaldo, Fecha Inicio, Fecha Fin, Rol)

[4] Fichero que se implementa a partir del resultado de la consulta del reporte en el SABIC. Le sirve como fuente de datos al diseño del reporte.

- ✓ Permite que el sistema Quarxo ejecute el reporte para cualquier juego de datos.

4. Cantidad de campos "*Field*" (**31 campos**)
   - ✓ Provoca que el diseño del reporte en SABIC demore mucho más ya que hay que colocar, organizar y enlazar un total de 62 campos y etiquetas.

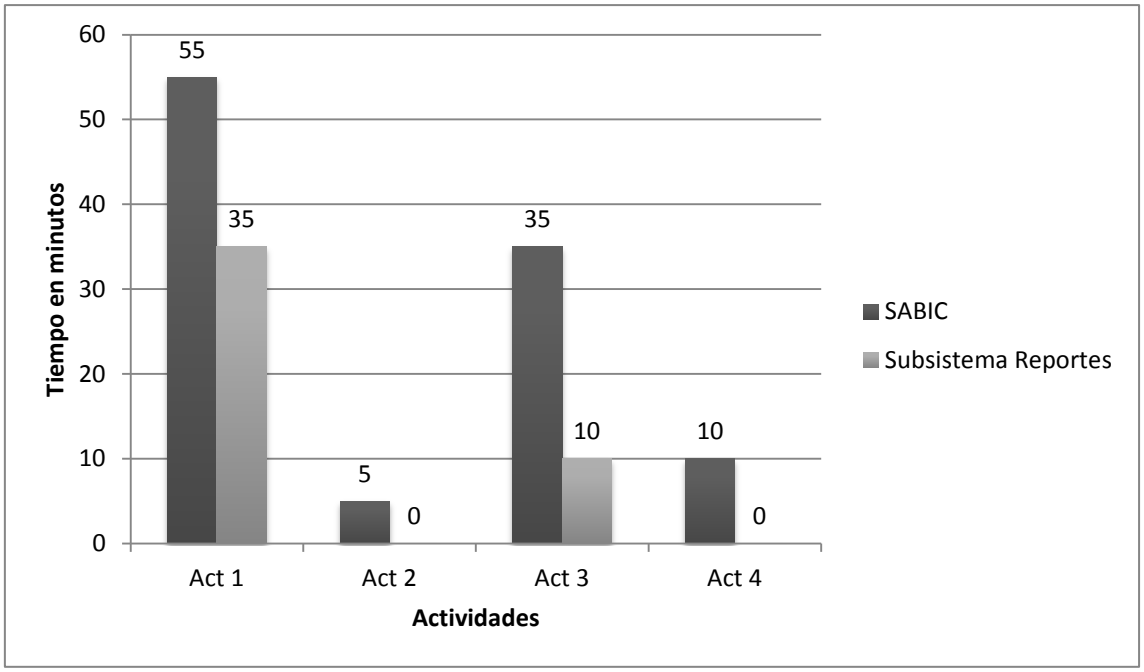


Figura 17. Comparación del tiempo de creación de un reporte por los sistemas SABIC y Quarxo.

A través del estudio de las actividades realizadas por ambos sistemas para la creación o modificación de un reporte, y su aplicación mediante el *Escenario 1*, se demuestra que con el subsistema Reportes se acorta el tiempo de realización de estas operaciones, a partir de la reducción del número de actividades y la disminución de la complejidad de las mismas.

3.2.3.2. Ahorro de recursos materiales

Contar con los recursos materiales adecuados y realizar un uso racional de los mismos es un elemento clave en la gestión de las organizaciones. Debido a los inconvenientes que presenta el SABIC, el 97 % de los reportes deben ser impresos para su posterior manejo. El continuo crecimiento de los reportes en el BNC provoca que el gasto de recursos en cuestiones de papel sea cada vez mayor.

El desarrollo del subsistema Reportes de Quarxo posibilita una mejor gestión de los reportes, más inmediata, cómoda y sobre todo, sin la necesidad de impresión;

permitiendo exportar a varios formatos convencionales (PDF, EXCEL) y consultar tantos reportes como sea necesario desde el sistema. Por tanto, solo habrá que imprimir aquellos documentos que por reglamentación necesiten ser archivados.

En la actualidad, el BNC genera elevados gastos monetarios en cuanto a materiales de impresión, debido a la gran cantidad de información que se maneja diariamente. La tabla a continuación, muestra los datos referentes al gasto de recursos relacionado con la gestión de reportes entre los sistemas SABIC y Quarxo, en el período correspondiente a los meses de marzo y abril del presente año.

	SABIC		Subsistema Reportes	
RECURSO	**CANTIDAD**	**PRECIO**	**CANTIDAD**	**PRECIO**
Papel Continuo	37 cajas	769.60 CUC	21 cajas	436.80 CUC
Papel “Bond”	56 paquetes	242.48 CUC	36 paquetes	155.88 CUC
TOTAL		**1012.08 CUC**		**592.68 CUC**

Tabla 7. Gasto de recursos relacionado con la gestión de reportes (marzo-abril).

Escenario 2:

Ejecución del reporte Estado de Cuentas: El reporte debe ser ejecutado una vez al mes con el objetivo de estructurar información de cada una de las empresas asociadas al BNC.

SABIC

1. Se ejecuta el reporte para cada una de las empresas.
    - ✓ Se imprime el resultado para cada una de las empresas.
    - ✓ Se destina a un trabajador para que atienda a las distintas empresas que vienen al BNC a recoger la información.

Quarxo

1. Se ejecuta el reporte automáticamente para cada una de las empresas de forma periódica de forma tal que:
    - ✓ Se le envíe un correo con la información pertinente a cada una de las empresas.
    - ✓ Se guarde una copia en la base de datos para constancia y respaldo de la información.

Se concluye que con lo explicado anteriormente quedan validadas las variables consideradas en la investigación, demostrándose que la solución desarrollada

contribuye a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

3.3. Conclusiones del capítulo

- ✓ Se obtuvo el subsistema Reportes de Quarxo, desarrollado con tecnologías libres, y provisto de elementos de seguridad, estándares y un entorno amigable, gracias a la utilización del *framework* Spring y la librería *JasperReports*.
- ✓ El subsistema fue probado mediante pruebas de Caja Blanca y Caja Negra, y validado utilizando pruebas de aceptación.
- ✓ Se evaluaron las variables consideradas en la investigación, demostrando que la solución desarrollada contribuye a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

CONCLUSIONES

La investigación desarrollada y los resultados obtenidos permiten a los autores plantear las siguientes conclusiones:

- ✓ En el estudio de los mecanismos para la obtención de reportes, se descartó la utilización de las aplicaciones informáticas ajustadas al negocio, por no corresponderse con la solución que se necesita. Se consideró *IReport* como herramienta especializada para la generación de los reportes.
- ✓ El proceso de desarrollo de software fue guiado por la metodología RUP, cumpliendo con los elementos que dicha metodología propone y logrando una efectiva ingeniería de software.
- ✓ Las tecnologías seleccionadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales.
- ✓ Se obtuvo el subsistema Reportes de Quarxo, probado mediante pruebas de caja Blanca y Caja Negra, y validado utilizando pruebas de aceptación. Fue desarrollado con tecnologías libres, y provisto de elementos de seguridad, estándares y un entorno amigable, gracias a la utilización del *framework Spring* y la librería *JasperReports*.

Por lo antes expuesto, se considera que la base teórica, la selección de las tecnologías para construir el subsistema Reportes de Quarxo y su implementación, son los principales aportes de este trabajo de diploma. La solución desarrollada contribuye a la gestión de reportes en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

RECOMENDACIONES

A partir del estudio realizado en la presente investigación, teniendo en cuenta las experiencias obtenidas a lo largo de su desarrollo, los autores proponen las siguientes recomendaciones:

- ✓ Desarrollar un mecanismo que permita insertar las plantillas de los reportes desde el subsistema.
- ✓ Adicionar la funcionalidad de consultar los reportes ejecutados en varios formatos.
- ✓ Estudiar la librería *JFreeChart* para la generación de gráficas que incluye soporte para: gráficas de pie, de barra, lineales, de intervalos de tiempo, y otros elementos que ayudan a la toma de decisiones. Su integración con *JasperReports* no implica gran complejidad.

BIBLIOGRAFÍA CONSULTADA

Design Patterns. Addison-Wesley, 1995. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Dojo: The Definitive Guide. O'Reilly Media Inc., 2008. Matthew A. Russell.

El Proceso Unificado De Desarrollo De Software. PERSON EDUCACIÓN.S.A. 2000. Ivan Jacobson, Grady Booch, James Rumbaugh.

INGENIERÍA DEL SOFTWARE. Un enfoque práctico. McGraw-Hill Interamericana, 2002. Roger S. Pressman.

IReport 3.7 Learn how to use *IReport* to create, design, format, and export reports. PACKT Publishing 2010. Shamsuddin Ahammad.

IText in Action. Oreilly & Associates Inc., 2010. Bruno Lowagie.

JasperReports 3.5 for Java Developers. PACKT Publishing, 2009. David R. Heffelfinger.

JUnit in Action. Manning Publications Co, 2010. Peter Tahchiev, Felipe Leme, Vincent Massol y Gary Gregory.

The Rational Unified Process: an introduction. Addison-Wesley Professional, 2004. Philippe Kruchten.

UML y Patrones. Prentice Hall, 1999. Craig Larman.

REFERENCIAS BIBLIOGRÁFICAS

1. *SQL Server 2008: Reporting*. [citado el 15 de nobiembre 2011]; Disponible en: http://www.microsoft.com/sqlserver/2008/en/us/reporting.aspx.
2. Gómez, A. y D. Fernando, *Diseño e implantación de un sistema de control administrativo/financiero de activos fijos para la Cruz Roja Ecuatoriana Quito.* 2010.
3. Rodríguez Durán, A. y otros, *IMPACTO DEL GENERADOR DINÁMICO DE REPORTES EN LOS SISTEMAS DE GESTIÓN DE INFORMACIÓN.* 2012.
4. *Unidad de Información y Análisis Financiero*. [citado el 11 de noviembre 2011]; Disponible en: http://www.uiaf.gov.co/?idcategoria=2626.
5. Iza Borja, G. *Sistema de Administración Financiera*. 2008 [citado el 17 de noviembre 2011]; Disponible en: https://esigef.mef.gov.ec/esigef/Ayuda/RUP-DS-037%20Instructivo%20de%20Reportes%20Din%C3%A1micos.pdf.
6. Sosa Porteiro, M. y P. Cobo Morales, *El VERSAT-Sarasola: Sistema cubano de Gestión Contable-Financiero.* 2006.
7. *Rodas XXI*. 2002 [citado el 19 de noviembre 2011]; Disponible en: http://www.rodasxxi.cu/.
8. *Professional Reporting for JAVA Applications*. 2011 [citado el 30 de abril 2012]; Disponible en: http://www.sap.com/solutions/sap-crystal-solutions/query-reporting-analysis/sapcrystalreports-eclipse/index.epx.
9. Xtras.Net. *ActiveX Report Designer for Visual Basic*. [citado el 23 de abril 2012]; Disponible en: http://www.xtras.net/products/activereports/.
10. Hidalgo, M. *Reporting Services* 2001 [citado el 13 de noviembre 2011]; Disponible en: http://www.sqlmax.com/reportin_services1.asp.
11. Ramanujam, S. y otros. *R2D: A bridge between the semantic web and relational visualization tools*. 2009: IEEE.
12. Colectivo de Autores, *PATDSI 2.0. Paquete de Herramientas para la Ayuda a la Toma de Decisiones.* 2009. **2.0**.
13. Shamsuddin, A., *iReport 3.7. Learn how to use iReport to create, design, format, and export reports.* 2010: PACKT Publishing. BIRMINGHAM - MUMBAI.
14. *Oxford Dictionaries. The world´s most trusted dictionaries.* 2012 [citado el 31 de mayo 2012]; Disponible en: http://oxforddictionaries.com/definition/library.
15. Heffelfinger, D., *JasperReports 3.5 for Java Developers*. 2009: Packt publishing.
16. Quintuña Churo, B.R. y V.C. García Alpala, *Sistema generador de reportes dinámicos para web, configurable para las plataformas de bases de datos más conocidas.* 2010.
17. Lowagie, B., *iText in Action*. 2010: Oreilly & Associates Inc.
18. Kruchten, P., *The rational unified process: an introduction*. 2004: Addison-Wesley Professional.
19. Larman, C., *UML y Patrones*. Segunda Edición ed. 1999: Prentice Hall.

20. Architecture Working Group, I., *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Standard IEEE Std 1471-2000.* IEEE Standards Association, 2000.

21. Marquina, E., *Guía de Patrones, Prácticas y Arquitectura .NET.* ORGANIZACIÓN CONTOSO, 2008.

22. ORACLE. *Core J2EE Patterns - Data Access Object*. 2009 [citado el 25 de abril 2012]; Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

23. Jaworski, J., *Java developer's guide*. Vol. 201. 1996: Sams. net.

24. Astarita, E. *Manual de XHTML*. [citado el 6 de diciembre 2011]; Disponible en: http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.

25. Díaz, F.J., C. Queiruga, y L.A. Fava. *Struts y JavaServer Faces, cara a cara*.

26. Milián, V. y otros, *PROPUESTA DE FRAMEWORK PARA SERVICIOS BASADOS EN LOCALIZACION EN CUBA.*

27. Iglesias Chaviano, A.M., *Modernización Sistema del Banco Nacional de Cuba.* Documento de Arquitectura, 2009.

28. Russell, M., *Dojo: the definitive guide*. 2008: O'Reilly Media, Inc.

29. Geer, D., *Eclipse becomes the dominant Java IDE.* Computer, 2005. **38**(7): p. 16-18.

30. Chopra, V., S. Li, y J. Genender, *Professional Apache Tomcat 6*. 2011: Wrox.

31. Elster, J., *Sour grapes: Studies in the subversion of rationality*. 1985: Cambridge Univ Pr.

32. Delaney, K., *Inside microsoft® sql server™ 2005: query tuning and optimization*. 2007: Microsoft Press.

33. Jacobson, I., G. Booch, y J. Rumbaugh, *EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE*. 2000: PERSON EDUCACIÓN.S.A.

34. Pressman, R.S., *INGENIERÍA DEL SOFTWARE. Un enfoque práctico*. Secta Edición. 2002: McGraw-Hill Interamericana.

35. Barnes, D.J. y M. Kölling, *Programación orientada a objetos con Java.* MANEJO EFICIENTE DEL TIEMPO, 2007.

36. Tahchiev, P. y otros, *Junit in action*. 2010: Manning Publications Co.

37. Myers, G., C. de Filecich, y A. Filevich, *El arte de probar el software*. 1983: El Ateneo.

GLOSARIO DE TÉRMINOS

A

AJAX: acrónimo de *Asynchronous JavaScript* And XML (*JavaScript* Asíncrono y XML). Es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones hacen posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. (**22**)

API: Interfaz de Programación de Aplicaciones o API (*Application Programming Interface*). Es el conjunto de funciones, procedimientos o métodos, en la programación orientada a objetos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción. (**18, 20, 22, 58**)

B

Beans: son componentes hechos en software que se pueden reutilizar y que pueden ser manipulados visualmente por una herramienta de programación en lenguaje Java. (**20**)

C

CASE: del inglés (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. (**15, 24**)

CSS: acrónimo de *Cascading Style Sheets* (Hojas de Estilo en Cascada), es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (*World Wide Web Consortium*) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. (**22**)

Código abierto: en inglés *Open Source*, es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código. (**10, 11, 12, 13, 14, 20, 22, 23**)

D

DOM: *Document Object Model* (en español, pero no oficialmente, Modelo en Objetos para la representación de documentos o Modelo de Objetos del Documento), es una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML. Es un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. (**22**)

E

Encapsulamiento: es el ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que solo se puede cambiar mediante las operaciones definidas para ese objeto. (**17, 18**)

Entidades: organización administrativa, comercial, económica, productiva y de servicios de carácter estatal, cooperativa, privada o mixta, residentes en el territorio nacional; así como las organizaciones sociales y de masas del país. (**1, 2, 3, 7, 8**)

H

HSQLDB: del inglés *Hyperthreaded Structured Query Language Database*. Es un sistema gestor de bases de datos libre escrito en Java. (**10**)

HQL: Del inglés *Hibernate Query Language*. Es un lenguaje consultor potente y muy parecido al SQL, completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. (**21, 22**)

I

IDE: Entorno de desarrollo integrado (en inglés *Integrated Development Environment*). Es un programa informático compuesto por un conjunto de herramientas de programación. (**15, 22, 23, 64**)

J

JavaBeans: son un modelo de componentes creado por *Sun Microsystems* para la construcción de aplicaciones en Java. (**11, 12**)

***JavaScript*:** es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. (**19, 22, 56**)

JDBC: *Java Database Connectivity*. Permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java. (**10, 11, 20**)

JFreeChart: es un marco de software de código abierto para el lenguaje de programación Java, el cual permite la creación de gráficos complejos de forma simple. (**11, 73**)

JNDI: Interfaz de Nombrado y Directorio Java (*Java Naming and Directory Interface*) es una Interfaz para servicios de directorio. Permite a los clientes descubrir y buscar objetos y nombres a través de un nombre. (**20**)

JRDataSource: es una interfaz que personifica la representación abstracta de un origen datos de *JasperReports*. Todos los tipos de fuente de datos deben implementar esta interfaz. (**12**)

M

Mapear: hace referencia a la acción de realizar un mapeo, (del inglés *mapping*), que significa hacer corresponder en elemento con otro en el sistema. (**20, 21, 66**)

MS-DOS: *Microsoft Disk Operating System*, Sistema operativo de disco de *Microsoft*. (**2, 69**)

Multiplataforma: es un término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas. (**7, 19, 22**)

Multivaluados: representa un conjunto de datos que puede tener un valor, varios o ninguno. Resulta muy útil en las funciones de parámetros. (**54, 55**)

P

Plugin: también conocido como (*plug-in, add-in, addon, add-on, snap-in o snapin*), es un pequeño programa de computadora que extiende las capacidades de otro programa de mayor tamaño, adicionándole nuevas funcionalidades. (**22**)

S

Script: conjunto de instrucciones escritas en un lenguaje *script* ejecutadas por un intérprete de comandos. (**22**)

T

TableModels: es un componente visual de java que nos permite dibujar una tabla, de forma que en cada fila/columna de la tabla podamos poner el dato que queramos. (**11**)

X

XHTML: siglas del inglés *eXtensible HyperText Markup Language*. Es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores entre otros. (**19, 22**)