Universidad de las Ciencias Informáticas
Facultad 4

Título:

Software para la Digitalización y Gestión Documental de los Registros y Notarías del MIJ de Venezuela.

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

Autores:

Yunior Martori Domenech.

René Queizán Pérez.

Tutor:

Alain Eduardo Rodriguez Arias.

La Habana, junio del 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al proyecto Registros y Notarías de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yunior Martori Domenech

René Queizán Pérez

_____

Alain Eduardo Rodriguez Arias

_____

DATOS DE CONTACTO

Tutor: Alain Eduardo Rodriguez Arias
Correo Electrónico: arod@uci.cu

Autor: Yunior Martori Domenech
Correo Electrónico: ymartori@estudiantes.uci.cu

Autor: René Queizán Pérez
Correo Electrónico: rqueizan@estudiantes.uci.cu

AGRADECIMIENTOS

Queremos dar nuestro más sincero agradecimiento a aquellos que contribuyeron a la realización de este trabajo, de forma directa e indirecta, en especial a nuestros padres, tutor Alain E. Rodriguez Arias y a Julio Cesar Díaz Vera.

Dar un agradecimiento especial a nuestro comandante, por darnos la oportunidad de estudiar en la Universidad de las Ciencias Informáticas, y poder realizar nuestros sueños.

DEDICATORIA

Dedicado a nuestros padres, amigos.

RESUMEN

El trabajo presenta una propuesta de solución de software para la digitalización y gestión documental de los documentos legales de los Registros y Notarías del MIJ de Venezuela. Se expone la plataforma sobre la cual fue desarrollado el sistema, las librerías usadas en la construcción del mismo, los componentes desarrollados. Se exponen además las técnicas de programación empleadas en el desarrollo del sistema.

PALABRAS CLAVES

- **Digitalización**
- **Gestión Documental**

TABLA DE CONTENIDOS

INTRODUCCION

Problema a resolver

Los Registros y Notarías del MIJ de Venezuela posee sus documentos legales solo en formato físico, debido a esto al realizar cualquier gestión u operación sobre los mismos es dificultoso, debido a varias razones:

Cualquier individuo que desee consultar sus documentos recibirá una copia de los originales, que se creará en dicho momento de forma manual, ocasionando pérdida de tiempo y consumo de recursos, papel, tinta, etc.

El individuo debe personarse en la misma oficina donde se registró, esto obliga a los ciudadanos a viajar a diferentes estados para revisar sus documentos.

La operación de búsqueda de los documentos se dificulta debido al alto número que existen, aún más que la búsqueda es de forma manual.

Los documentos en su mayoría son muy antiguos, por lo que su estado es crítico y se van dañando a medida que se van usando, existiendo el riesgo de dañarse y perder su valor legal.

Actualidad y necesidad del trabajo

Con el objeto de eliminar y dar solución a todas las dificultades antes mencionadas, además de la necesidad de modernizar los Registros y Notarías del MIJ de Venezuela se presentó esta propuesta de solución.

El presente trabajo presenta una propuesta de solución de Software para la Digitalización y Gestión Documental de los documentos legales del MIJ de Venezuela. Dicha solución permitirá la digitalización de los documentos, clasificación de los documentos en Unidades Documentales, almacenar en soporte digital los mismos, gestionar cualquier operación sobre los documentos.

El sistema almacenara las trazas de todas las operaciones realizadas en el sistema:

- Quién realizó la operación.

- Cuando fue realizada.
- Qué tipo de operación realizo.
- El puesto de trabajo donde la realizó.

La solución facilitará la búsqueda y consulta de los documentos de forma rápida y eficiente, mostrando el documento en el ordenador, de ser necesario se imprimirá, se podrán gestionar desde cualquier estado; independientemente del estado del documento, el número de consultas que se le realicen no dañará más el documento, ya que se trabaja con la copia digital ya realizada.

Aportes prácticos esperados del trabajo

El sistema facilita la digitalización de tomos para el proyecto Registros y Notarías de Venezuela a lo que se suma un control de la calidad de escaneo de los mismos y la obtención de reportes acerca de la productividad de escaneo. Permite, además, almacenar el historial de todas las operaciones que se realizan sobre estos documentos oficiales de forma tal que favorezca el control estricto de las alteraciones que se lleven a cabo.

Objeto de estudio

Proceso de digitalización y gestión de los tomos existentes en los Registros y Notarías del MIJ de Venezuela.

Campo de acción

Proceso de digitalización y almacenamiento de los documentos de los Registros y Notarías del MIJ de Venezuela, clasificación de los tomos en Unidades Documentales y exportación de las mismas SAREN1.

[1] Servicio Autónomo de Registros y Notarías

Objetivos generales

- Desarrollar una aplicación que permita controlar el proceso de digitalización y gestión documental.

Objetivos específicos

- Personalizar el componente de escaneo que brinda la LeadTool.
- Desarrollar el seguimiento de los documentos por cada área, usuario, operación.
- Controlar la calidad de las imágenes escaneadas.
- Integrar con el software de Registro y Notaria.

Tareas desarrolladas para cumplir los objetivos

- Estudio de Microsoft Visual Studio.Net 2003 como entorno de desarrollo para la aplicación.
- Estudio de NHibernate como gestor de acceso a datos.
- Estudio de Oracle y Pl/Sql como base de datos y gestor de base de datos a usar.
- Estudio de la LeadTools, librerías que permitirán el proceso de escaneo.
- Estudio de Crystal Report para la creación de reportes.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El presente capítulo está destinado a brindar una breve descripción de las principales técnicas de programación usadas en la actualidad, además de realizar un análisis de un conjunto de tecnologías, enfatizando las utilizadas en el desarrollo de este trabajo.

1.2 Paradigmas de Programación

1.2.1 Programación Estructurada

Programación Estructurada es una técnica en la cual la estructura de un programa, se realiza tan claramente cómo es posible mediante el uso de tres estructuras lógicas de control:

a. **Secuencia**: Sucesión simple de dos o más operaciones.
b. **Selección**: bifurcación condicional de una o más operaciones.
c. **Interacción**: Repetición de una operación mientras se cumple una condición.

Las estructuras básicas en cualquier lenguaje de programación son: los bloques, las estructuras secuenciales, las estructuras repetitivas (*Repeat/Until*, *Do/While*, *For/Do*) las estructuras condicionales o selectivas (*If/Then/Else*, I*f/Then*, *Case/Of*). La presencia de instrucciones de tipo *Go/To* contradice las metodologías de la programación fundamentales y sólo se justifica en los métodos primitivos basados en los diagramas de flujo.

El principal inconveniente de este método de programación, es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático su manejo, esto se resuelve empleando la programación modular, definiendo módulos interdependientes programados y compilados por separado. Un método un poco más sofisticado es la programación por capas, en la que los módulos tienen una estructura jerárquica muy definida. (1)

Ventajas de la programación estructurada:

a. Se evita la repetición de trabajo.
b. Trabajo de programación compartimentado en módulos independientes.
c. Diseño top-down: descomposición en subproblemas.
d. Facilita el mantenimiento.
e. Legibilidad.
f. Independencia de los módulos.
g. Favorece la reutilización.

1.2.2 Programación Lógica

El paradigma lógico se basa en la idea que un problema puede ser descrito definiendo relaciones entre los objetos de un determinado conjunto, y que a partir de éstas, otras relaciones pueden ser calculadas a partir de una determinada regla de inferencia. La lógica formal provee el basamento para este paradigma.

La denominada Programación Lógica, que se caracteriza por la utilización de un subconjunto de la Lógica de Primer Orden (las cláusulas de Horn) como lenguaje para la representación del conocimiento.

1.2.3 Programación Orientada a Objetos

La tecnología orientada a objetos se apoya en los sólidos fundamentos de la ingeniería, cuyos elementos reciben el nombre global de modelo de objetos. El modelo de objetos abarca los principios de abstracción, encapsulación, modularidad, jerarquía, tipos, concurrencia y persistencia. Ninguno de estos principios es nuevo por sí mismo. Lo importante del modelo de objetos es el hecho de conjugar todos estos elementos de forma sinérgica.

La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una

instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia.

La programación orientada a objetos ayuda a los desarrolladores a crear objetos que reflejan escenarios del mundo real, las implementaciones orientadas a objetos ocultan datos, lo cual significa que muestran únicamente los comportamientos a los usuarios y ocultan el código subyacente de un objeto. Los comportamientos que el programador expone son únicamente aquellos elementos que el usuario de un objeto puede afectar.

Elementos fundamentales del modelo de objeto:

Abstracción: La abstracción es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales).

Encapsulamiento: Es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales.

Modularidad: La Modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas llamadas módulos, cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

Jerarquía: La Jerarquía es una propiedad que permite la ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (jerarquía “es-un”: generalización/especialización) y estructura de objetos (jerarquía “parte-de”: agregación).

Polimorfismo: Es la propiedad que indica, literalmente, la posibilidad de que una entidad tome muchas formas.

El modelo objeto ideal no sólo tiene las propiedades anteriormente citadas sino que es conveniente que soporte, además, estas otras propiedades:

- ***Concurrencia*** (multitarea): el lenguaje soporta la creación de procesos paralelos independientes del sistema operativo.
- ***Persistencia:*** los objetos deben poder ser persistentes; es decir, los objetos han de poder permanecer después de la ejecución del programa
- ***Genericidad:*** las clases parametrizadas (mediante plantillas o unidades genéricas) sirven para soportar un alto grado de reutilización. Estos elementos genéricos se diseñan con parámetros formales, que se instanciarán con parámetros reales, para crear instancias de módulos que se compilan y enlazan, y ejecutan posteriormente.
- ***Manejo de Excepciones:*** se deben poder detectar, informar y manejar condiciones excepcionales utilizando construcciones del lenguaje. Esta propiedad añadida al soporte de tolerancia a fallos del software permitirá una estrategia de diseño eficiente.
- ***Tipificación:*** Un tipo es una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades.

La programación orientada a objetos nos brinda un sinnúmero de ventajas, la uniformidad a la hora de representar los objetos del análisis, diseño y la codificación de los mismos, la flexibilidad, la estabilidad, la reusabilidad y la mantenibilidad entre otras.

No hay un estilo de programación que sea el mejor para todo tipo de aplicaciones. Por ejemplo, la programación orientada a reglas sería la mejor para el diseño de una base de conocimiento, y la programación orientada a procedimientos sería la más indicada para el diseño de operaciones de cálculo intensivo. Por nuestra experiencia, el estilo orientado a objetos es el más adecuado para el más amplio conjunto de aplicaciones; realmente, este paradigma de programación sirve con frecuencia como el marco de referencia arquitectónico en el que se emplean otros paradigmas. (2)

1.2.4 Programación Orienta a Aspectos

En el desarrollo de un sistema software además del diseño y la implementación de la funcionalidad básica, se recogen otros aspectos tales como la sincronización, la distribución, el manejo de errores, la optimización de la memoria, la gestión de seguridad, etc. Con las descomposiciones funcional y orientada a objetos no se aíslan bien estos aspectos, sino que quedan diseminados por todo el sistema enmarañando el código que implementa la funcionalidad básica, y yendo en contra de la claridad del mismo.

Debido a esto surge la programación orientada a aspectos (POA) que es una nueva metodología de programación que aspira a soportar la separación de incumbencias para los aspectos antes mencionados. Es decir, que intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema. En definitiva, lo que se persigue es implementar una aplicación de forma eficiente y fácil de entender.

¿Qué es un aspecto?

La definición de aspecto ha evolucionado a lo largo del tiempo, pero con la que se trabaja actualmente es la siguiente:

Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa (G. Kiczales).

Un ejemplo clásico de la programación orientada a aspectos es el registro de la actividad de una aplicación. En varios puntos de ejecución, queremos invocar un método que registra en un archivo de texto o en una base de datos el hecho de que se ha llegado a ese punto (una traza del sistema).

Las consecuencias directas de estas incumbencias transversales son el código disperso y enredado.

Código enredado: una clase o módulo, además de implementar su funcionalidad principal debe ocuparse de otras competencias o intereses.

Código disperso: el código que satisface una competencia está esparcido por distintas partes del sistema. Podemos distinguir dos tipos de código disperso:

- *Bloques de código duplicados*: cuando los mismos bloques de código aparecen en distintas partes del sistema.
- *Bloques de código complementarios*: cuando las distintas partes de una incumbencia son implementadas por módulos diferentes.

El código disperso y enredado es un código difícil de reutilizar, mantener y evolucionar y de pobre trazabilidad. Estos efectos hacen que disminuya la calidad del software diseñado y se reduzca la productividad, sin embargo el paradigma POA permite capturar las **incumbencias transversales**[2] que atraviesan el sistema en entidades bien definidas llamadas **aspectos**[3], consiguiendo la separación de los intereses, eliminando el código disperso y enredado, y con ello todos los efectos negativos que éste implica. Debemos resaltar que la programación orientada a aspectos no sustituye al paradigma en el que se basa la construcción del sistema, sino que está un nivel de abstracción por encima. (3) (4)

En el sistema se uso la POO pues facilita el entendimiento del negocio al poder modelar el sistema con conceptos que pertenecen al campo de acción del problema, además de que brinda todos los beneficios de la programación orientada a objetos.

[2] Conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema, o varias partes de él.
[3] Concepto que no es posible encapsular, y por tanto, resulta diseminado por todo el código.

1.3 Plataformas de desarrollo

1.3.1 Java

Fue desarrollado por Sun Microsystems y es un lenguaje de programación orientado a objetos que es capaz de generar aplicaciones independientes, y puede ser utilizado en Aplicaciones en Servidor así como en Aplicaciones en Cliente, y otra gran gamma de aplicaciones.

- Basado en C++ pero simplificado, mucho más fácil de usar, de más alto nivel y menos propenso a errores.
- Amplísima biblioteca estándar de clases predefinidas.
- Las aplicaciones Java pueden ser ejecutadas indistintamente en cualquier plataforma sin necesidad de recompilación.
- Gestión avanzada de memoria mediante un recolector de basura.
- Gestión avanzada de errores, tanto en tiempo de compilación como de ejecución.
- Distribuido y multihilo.

JDK o SDK:

JDK es el ambiente en el cual es posible desarrollar cualquier aplicación Java. Este ambiente o paquete incluye: El API de Java, el compilador de Java que convierte código fuente de Java en el bytecode de Java, así como el JVM "Java Virtual Machine" de la plataforma correspondiente.

JRE (Entorno en tiempo de ejecución para Java):

Este ambiente es utilizado solo para ejecutar programas en Java. Contiene la JVM que proporciona Sun Microsystems, junto con su implementación de las librerías estándar. El JRE es lo mínimo que debe contener un sistema para poder ejecutar una aplicación Java sobre el mismo.

API de Java:

Estas librarías estándar, ofrecen al programador un conjunto bien definido de funciones para realizar tareas comunes Además, las librerías proporcionan una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su sistema operativo. Tareas tales como manejo de las funciones de red o acceso a ficheros, suelen depender fuertemente de la funcionalidad nativa de la plataforma destino.

Máquina Virtual de Java:

Ejecuta el código resultante de la compilación del código fuente, conocido como bytecode. Se encarga de traducir el bytecode en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre que con una implementación adecuada de la JVM.

Ventajas de este sistema:

- Se compila la aplicación una única vez y los ejecutables en bytecode obtenidos son válidos para cualquier plataforma.
- El código fuente queda a salvo
- Es muy robusto. La máquina virtual Java es capaz de detectar y notificar gran cantidad de errores durante la ejecución de la aplicación.
- El recolector de basura no ocupa espacio en el ejecutable, ya que viene integrado en la JVM.
- Los ejecutables son pequeños porque las librerías de clases vienen proporcionadas junto a la JVM de la plataforma concreta.

Inconvenientes:

Evidentemente la interpretación o incluso compilación (just in time) del bytecode produce aplicaciones más lentas que en el caso de la ejecución directa de un binario. El recolector de basura puede suponer una sobrecarga adicional al procesador. (5)

1.3.2 FrameWork .NET

NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. Basado en esta plataforma, Microsoft intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el Sistema Operativo hasta las herramientas de mercado.

A largo plazo Microsoft pretende reemplazar el API Win32 o Windows API con la plataforma .NET. Esto debido a que el API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows.

NET intenta ofrecer una manera rápida y económica pero a la vez segura y robusta de desarrollar aplicaciones permitiendo a su vez una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

El framework o marco de trabajo constituye la base de la plataforma .NET y denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido.

Figura 1.1 Diagrama detallado del **Marco de Trabajo .NET.**

El CLR es el verdadero núcleo del Framework de .NET, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo (W2k y W2003).

Figura 1.2 Entorno de Común de Ejecución para Lenguajes (CLR)

El CLR es el verdadero núcleo del Framework de .NET, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes. Es responsable de los servicios en tiempo de ejecución como la integración de lenguajes, la aplicación de seguridad y la

administración de la memoria, los procesos y los subprocesos. Además, juega un importante papel en tiempo de desarrollo, puesto que características como la administración de la duración, la aplicación de nombres de tipos seguros, el control de excepciones entre lenguajes, la creación de enlaces dinámicos, etc.

El CLR tiene una componente denominada JIT que se encarga de ir convirtiendo dinámicamente el código MSIL a ejecutar en código nativo según sea necesario, por tanto actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET.

La actuación de JIT durante la ejecución de una aplicación gestionada puede dar la sensación de hacer que ésta se ejecute más lentamente debido a que ha de invertirse tiempo en las compilaciones dinámicas. Esto es cierto, pero hay que tener en cuenta que es una solución mucho más eficiente que la usada en otras plataformas como Java, ya que en .NET cada código no es interpretado cada vez que se ejecuta sino que sólo es compilado la primera vez que se llama al método al que pertenece. (6)

Bibliotecas de clases (BCL):

La Librería de Clase Base (BCL) es una librería incluida en el .NET Framework formada por cientos de tipos de datos que permiten acceder a los servicios ofrecidos por el CLR y a las funcionalidades más frecuentemente usadas a la hora de escribir programas. Además, a partir de estas clases prefabricadas el programador puede crear nuevas clases que mediante herencia extiendan su funcionalidad y se integren a la perfección con el resto de clases de la BCL.

La solución propuesta está basada en .NET debido a que proporciona una mayor velocidad de desarrollo al no tener que desarrollar detalles de infraestructura.

Ejemplo .NET Pet Shop, la versión basada en .NET de la aplicación de ejemplo de prácticas recomendadas de Sun, Java Pet Store, implementa la misma funcionalidad que la versión Java 2 Enterprise Edition (J2EE), pero utiliza un tercio del código de esta versión.

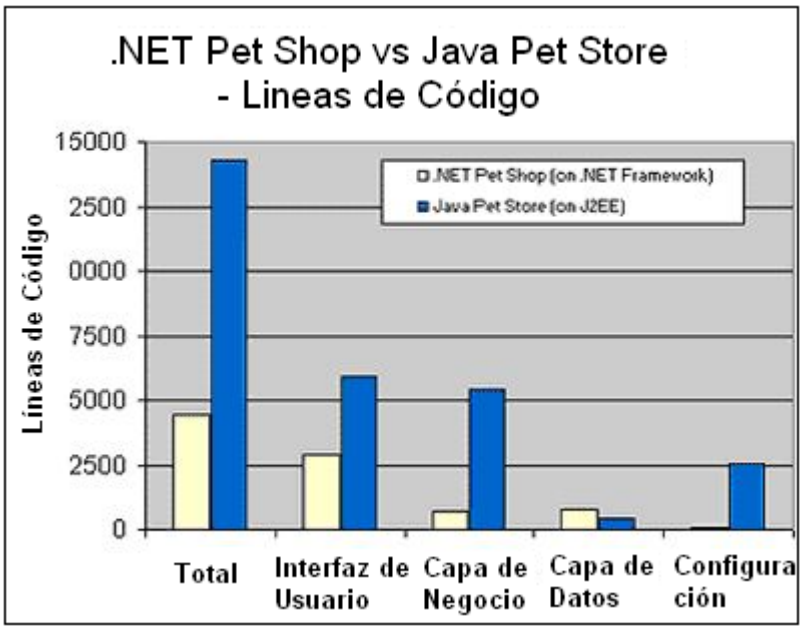

Figura 1.3 NET Pet Shop vs Java Pet Store.

La herramientas de desarrollo es Visual Studio® .NET 2003, que proporcionan un entorno de programación integrado de gran rapidez para la programación con .NET Framework. (7)

1.4 Leadtools

La Leadtools posee 16 años de experiencia en el desarrollo de herramientas de programación de software para la manipulación de imágenes.

Poseen interfaces de programación para diversos lenguajes: **.NET, C++, Delphi, ActiveX (www), COM (Component Object Model, OLE, Object Linking Embedding), API(Application Program Interface).**

Incluye además varios módulos:

- Barcode SDK pluggins.
- ICR SDK Module (Intelligent Character Recognition, convertir un reporte hecho a mano en un Pdf, Word).
- OCR SDK Module (Optical Character Recognition, reconocer textos en zonas de las imágenes de forma automática).
- OMR SDK Module (Optical Mark Recognition, para capturar las marcas de verificación de los documentos, usado en los exámenes, cuestionarios).
- TWAIN Scanning/Capture (provee un grupo de herramientas que se expande en el amplio espectro de las imágenes digitales, con un fuerte soporte a la adquisición de imágenes a través de las variadas interfaces estándares de dispositivos de hardware, se pueden controlar cámaras, escáneres o tarjetas de captura con los controladores de dispositivos Twain, fácil de usar, funcionalidad de bajo nivel y flexible).
- Raster Imaging Pro SDK (provee soporte para conversión de colores, algoritmos de compresión de imágenes, procesamiento de imágenes, efectos especiales, más de 2000, soporte a más de 150 tipos de archivos de imágenes). (8)

1.5 NHibernate

NHibernate 1.1 es un framework de Object-Relational-Mapping(ORM) open-source que resuelve de forma transparente la persistencia de objetos los objetos de contra una base de datos relacional. NHibernate está basado en el popular framework open-source Hibernate surgido en la comunidad Java en el año 2002.

Utilizar NHibernate ofrece las siguientes ventajas:

- Persistencia transparente: Mis objetos del dominio no saben nada acerca de la base de datos donde son persistidos, el framework lo resuelve en forma automática utilizando archivos de mapping expresados en XML.

- Soporte de polimorfismo: Puedo cargar jerarquías de objetos en forma polimórfica.
- Soporte de los 3 niveles de mapeo de herencia: Puedo mapear toda una jerarquía de clases a una sola tabla, crear una tabla por cada clase concreta o crear una tabla por cada escalón de la jerarquía.
- Soporte completo de asociaciones: NHibernate soporta el mapeo de todos los tipos de relaciones que pueden existir en un modelo de objetos del dominio (asociaciones 1..1, 1...N, N..M, unidireccionales y bidireccionales).
- Soporte de carga de objetos Proxy: Puedo cargar objetos que solo contengan la clave del objeto completo.
- Soporte de caching: En el contexto de una transacción, puedo disminuir la cantidad de veces que voy contra la base de datos cacheando en memoria los objetos que son accedidos varias veces.
- Soporte de múltiples dialectos SQL: Puedo independizarme completamente del tipo de base de datos utilizada. Mi aplicación puede persistir sus datos en SQL Server, en Oracle, en Postgres, etc. simplemente cambiando la configuración correspondiente.

Utilizar el framework NHibernate simplifica enormemente la programación de lógica de persistencia. (9)

1.6 NUnit

Esta herramienta de pruebas para .NET se encuentra en su segunda versión. Enteramente escrita en C#, ha sido rediseñada para tomar ventaja de muchas de las características de los lenguajes .NET, por ejemplo atributos personalizados y otras capacidades relacionadas con la reflexión. Además provee una interfaz grafica para ejecutar y administrar las mismas. Hay que resaltar que este framework es de distribución libre.

NUnit se encarga de analizar ensamblados generados por .NET, interpretar las pruebas inmersas en ellos y ejecutarlas. Utiliza atributos personalizados para interpretar las pruebas y provee

además métodos para implementarlas. En general, NUnit compara valores esperados y valores generados, si estos son diferentes la prueba no pasa, caso contrario la prueba es exitosa.

NUnit que puedes hacer debug asociando NUnit al proceso de depuración de Visual Studio, de esta manera es posible ejecutar las pruebas y hacer el seguimiento al programa al mismo tiempo.

NUnit basa su funcionamiento en dos aspectos, el primero es la utilización de atributos personalizados. Estos atributos le indican al framework de NUnit que debe hacer con determinado método o clase, es decir, estos atributos le indican a NUnit como interpretar y ejecutar las pruebas implementadas en el método o clase.

El segundo son las aserciones, que no son más que métodos del framework de NUnit utilizados para comprobar y comparar valores.
Los casos de prueba realizados sobre Nunit no son más que programas .NET, que quedan archivados y pueden ser re ejecutados tantas veces como sea necesario, de ahí su utilidad.

1.7 Crystal Reports

Crystal Reports es el generador de reporte por excelencia de Visual Studio.NET. Esta herramienta permite crear contenido interactivo con calidad de presentación, punto fuerte de Crystal Reports durante años, en la plataforma .NET.

Brinda la posibilidad de incluir gráficos de barra, de pastel; diseñar los reportes con una excelente calidad de presentación.Se puede construir los reportes en tiempo de diseño, mediante el diseñador de Crystal Reports de una forma muy fácil, rápida y eficiente.

Crystal Reports puede conectarse con casi cualquier BD incluyendo relacionales, OLAP y XML. Crystal Reports también soporta Unicode, así que puedes mostrar los datos almacenados virtualmente en cualquier lenguaje. Algunos ejemplos de las fuentes de datos soportadas

mediante los drivers nativos DB OLE, ODBC, y JDBC son: Oracle ® , IBM ® DB2 ® , Sybase, Microsoft ® SQL Server, Microsoft SQL Server Analysis Services, Informix ® , Act! TM , Pervasive, Pervasive SQL, Microsoft Access, Teradata, Holos, SAP BW, BDE, CDO, Microsoft Internet Information Server, Microsoft System Management Server, Microsoft Windows NT Event Logs, Microsoft Outlook, Paradox, XML/XSD, Lotus Domino, Microsoft FoxPro, Web Server Activity Logs (NCSA format), ASCII, Dbase, and Informix.

1.8 Componente Desarrollado

Para llevar a cabo el proyecto necesitamos escanear documentos, realizar una serie de operaciones para tratar los mismos, como rotarlos, eliminar los bordes, eliminar, insertar y adicionar páginas, para poder darle solución a este problema fue necesario la creación de un nuevo componente, que se encargara de digitalizar y tratar las imágenes, este componente se desarrollo reutilizando las librerías de la Leadtools.

Se reutilizaron los módulos TWAIN y Raster Imagin Pro SDK de la Leadtools, los cuales dan soporte al escaneo y tratamiento de las imágenes (formato de la imagen, colores, compresión, rotación, etc).

El componente se puede definir como una interfaz diseñada a nuestras necesidades específicas, reutilizando las funcionalidades que brinda la Leadtools, de forma tal que no existió la necesidad de implementar las funcionalidades que nos brindan las librería, solo reutilizarlas, lo cual ahorro un considerable tiempo de desarrollo de la aplicación, además recordemos que esta compañía lleva 16 años de experiencia en el desarrollo de herramientas de programación de software para la manipulación de imágenes.

Nuestro componente fue diseñado a la medida, solo brindamos las funcionalidades requeridas por nuestra aplicación. Dependiendo del rol que juega el componente en cada módulo del proyecto se le crearon varios roles para limitar las funcionalidades del componente:

- **Escanear:** Permite realizar las operaciones relacionadas con el escaneo, adición, y eliminación de páginas.
- **Calidad:** Permite realizar las operaciones relacionadas con la revisión, y limpieza de las imágenes, enderezarlas, eliminar bordes.
- **EscanerCalidad:** Permite realizar todas las operaciones.
- **Navegador:** Solo permite recorrer las imágenes.
- **NavegadorBásico:** Solo permite recorrer las imágenes moviéndose de una en una.
- **Visor:** Visualiza la imagen, pero no admite ninguna operación sobre la misma.

Figura 1.4 Menú del componente de escaneo desarrollado.

Las clases principales que se utilizaron de las librerías fueron:

- **RasterCodecs:** Permite fijar la compresión de la imagen, cargar, salvar, eliminar paginas.
- **TwainSession:** Permite escanear el documento, además de configurar una serie de parámetros como la escala, márgenes.
- **CodecsImageInfo:** Contiene toda la información perteneciente a la imagen que se está manipulando, cantidad de páginas, resolución, BPP, compresión, etc.
- **RasterImageViewer:** Es el componente que permite la visualización de la imagen.

1.9 Conclusiones

Luego de haber investigado sobre las diferentes plataformas de desarrollo, las técnicas de programación así como un conjunto de herramientas que facilitan la implementación del sistema, se pudo concluir que usando .Net con C# se puede brindar al cliente una rápida y eficiente respuesta a sus requerimientos.

Capitulo 2. Descripción y análisis de la solución propuesta

2.1 Introducción

En este capítulo abordamos la arquitectura del sistema, los patrones usados en la construcción del mismo, el tratamiento de errores, las estrategias de integración empleadas y las pautas de programación utilizadas.

2.2 Estrategias de Integración

Como ya se conoce nuestro objetivo es Dotar a los Registros y Notarías del MIJ de Venezuela de un Repositorio Digital de imágenes que contenga los documentos digitalizados y clasificados en UD, garantizando la disponibilidad de la información generada a los mismo, para ello fue necesario definir determinados convenios con el fin de realizar la integración de ambos proyectos.

Para esto se realizo el caso de uso ExportarTomo, cuyo objetivo es exportar al SAREN el resultado final del proceso de digitalización, que son las Unidades Documentales en formato TIFF junto con toda la información referente a ellas en un XML. Las direcciones de ambos módulos habían definido que la exportación se llevaría a cabo a través del siguiente sistema de carpetas.

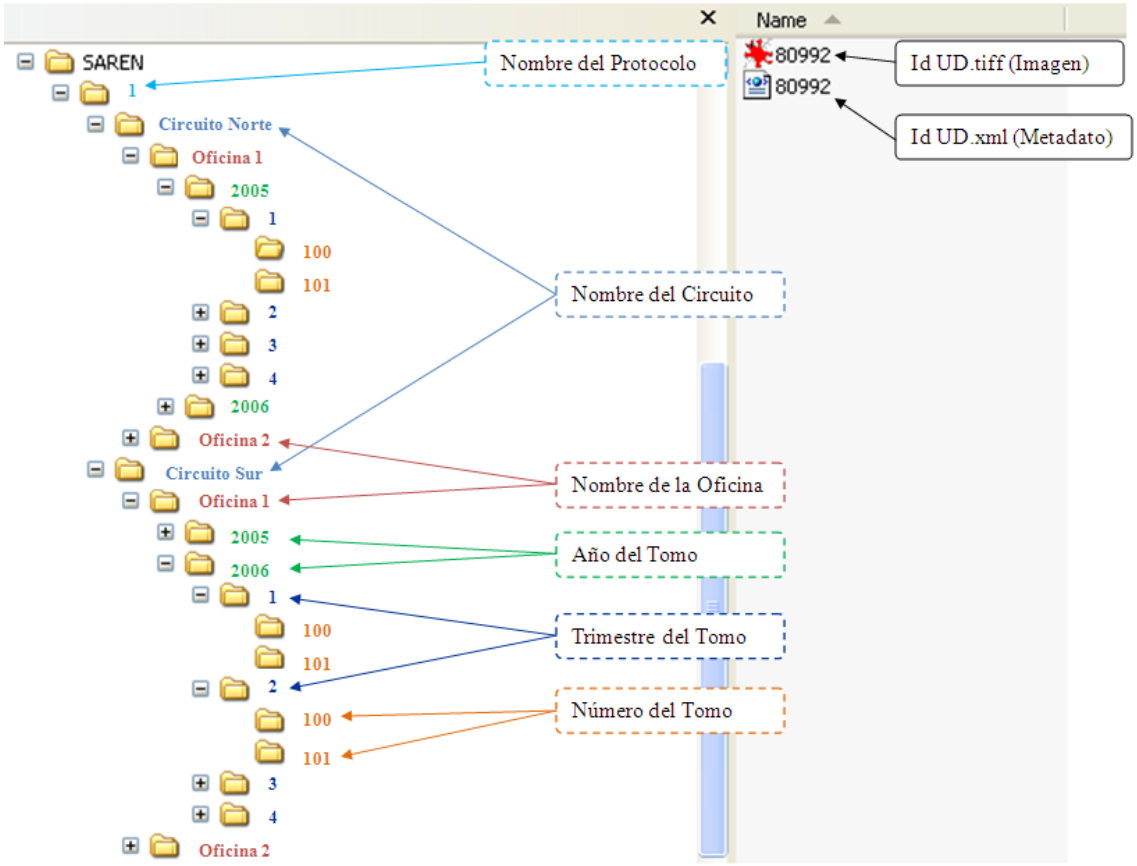

Figura 2.1 Estructura de carpetas de las Unidades Documentales exportadas.

Primeramente se creara una carpeta que represente el protocolo al que pertenece el Tomo que contiene esta unidad documental, dentro de esta se encontrara otra carpeta con los circuitos, un nivel más abajo se encuentran los nombres de las oficinas, dentro, el año de los tomos, los trimestres y los números de los tomos, luego se encuentran directamente las Unidades Documentales, formadas por dos archivos, TIFF que representa la imagen y XML que representa la metadato asociada a dicha UD.

De esta forma se crea una independencia total entre ambos sistemas ya que no interactúan entre sí, sino, tan solo con este sistema de carpetas. De esta forma cuando se realice un cambio en el diseño, en la representación de la base de datos o en la codificación del centro de

digitalización no tendrá ninguna repercusión en las oficinas de registro de notarias pues lo único que necesitan es la correcta estructura de carpetas.

2.3 Patrones y Arquitectura

Los patrones de diseño, describen un problema que ocurre repetidas veces en algún contexto determinado del desarrollo de software y entregan una buena solución ya probada. Esto ayuda a diseñar correctamente en menos tiempo, ayuda a construir soluciones reutilizables y extensibles, y facilita la documentación. Los patrones nos dicen cómo aplicar de manera eficaz la herencia, el polimorfismo y todas las ventajas que posee el Paradigma Orientado a Objetos. (10)

La mayoría de las aplicaciones distribuidas presentan la arquitectura de 3 capas esta se centra en:

1. La capa de presentación que en este caso está formada por los Componentes de IU, y los componentes de proceso de IU. Los componentes de IU pueden ser vistos como la parte con la cual interactuar el usuario, las ventanas, por decirlo de alguna manera. Los componentes de proceso de IU son asociados a clases controladoras. Es decir estos encapsulan lógica de navegación y control de eventos de la interface.
2. La capa de negocios encapsula los objetos que van a ser manejados o consumidos por toda la aplicación, y que representan la lógica del negocio.
3. La capa de acceso a datos que contiene clases que interactúan con la base de datos. Estas clases surgen como una necesidad de mantener la cohesión o clases altamente especializadas que ayuden a reducir la dependencia entre las clases y capas. Aquí podemos encontrar también una clase con métodos estáticos que permiten agrupar las operaciones de acceso a datos en una única clase denominada CADatosFachada.

A continuación se muestra la estructura de capas que se ha creado para reflejar la implementación de la aplicación:

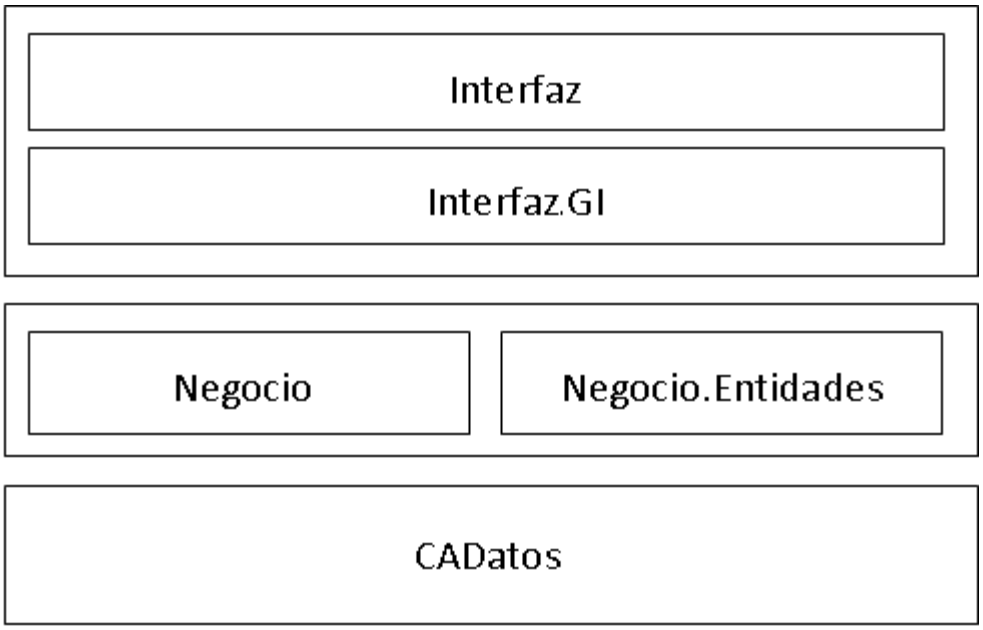

Figura 2.2 Arquitectura del Sistema.

Capa de presentación, contiene la declaración de todos los componentes de la IU, estos se encuentran implementados en CentroDigitalizacion.Interfaz.
La logica de presentación se implementa en el componente CentroDigitalizacion.Interfaz.GI, y es el encargado de implementar la lógica de navegación en la IU, lo cual permite una mayor limpieza en el código al separar las declaraciones de la implementación.

Capa de Negocios, aquí tenemos inicialmente a las entidades empresariales, que se implementaron en el proyecto CentroDigitalizacion.Negocio.Entidades, las cuales solo contienen los atributos con sus propiedades como se muestra a continuación:

```
public class ImagenTomo
{
    private int idTomo;
    private Entidades.Tomo.Tomo tomos;
    private byte[] imagen;
    private byte[] llave;
    public ImagenTomo()
    {
    }
    public ImagenTomo(int idTomo, byte[] imagen, byte[] llave)
    {
        this.idTomo = idTomo;
        this.imagen = imagen;
        this.llave = llave;
    }
    public int IdTomo
    {
        get
        {
            return idTomo;
        }
        set
        {
            idTomo = value;
        }
    }
    public Entidades.Tomo.Tomo TomoOb
    {
        get
        {
            return tomos;
        }
        set
        {
            tomos = value;
        }
    }
    public byte[] Imagen
    {
        get
        {
            return imagen;
        }
        set
        {
            imagen = value;
        }
    }
    public byte[] Llave
    {
        get
        {
            return llave;
        }
        set
        {
            llave = value;
        }
    }
}
```

Figura 2.3 Entidad.

Una clase ImagenTomo que está compuesta por un IdTomo que sirve como identificador, un objeto TomoOb que contiene todos los datos de un tomo, una Imagen almacenada en un arreglo de byte y una Llave que representan un Hash de la imagen.

La lógica de los componentes empresariales según la Arquitectura. Estará implementada por el componente CentroDigitalizacion.Negocio, y el objetivo es que contenga la lógica de negocios.

La capa de acceso a datos es implementado por el componente CentroDigitalizacion.CADatos, que contiene las clases CADatosFachada, NhibernateUtil y DaoGenerico.La clase NhibernateUtil posee una serie de métodos estáticos que se encargan del manejo de secciones y transacciones de Nhibernate, y la clase DaoGenerico que aumentan el nivel de reusabilidad de la lógica de acceso a datos, pues generaliza los procesos básicos de entrada y salida en la BD debido a que no necesita conocer el tipo de la clase para hacer cualquier operación sobre ella ya sea (insertar, borrar, seleccionar o actualizar).

En la propuesta de solución se creó la clase DaoGenerico que es la encargada de todos los accesos a los datos sin necesidad de conocer el tipo de dato con el que está operando, en la siguiente figura podemos apreciar un ejemplo:

```
public class DaoGenerico
{
    public DaoGenerico()
    {
    }
    public  static void Salvar(Object instance)
    {
        try
        {
            NHibernateUtil.getSession().SaveOrUpdateCopy(instance);
        }
        catch (Exception re)
        {
            throw re;
        }
    }
    public static void Eliminar(Object Instance)
    {
        try
        {
            NHibernateUtil.getSession().Delete(Instance);
        }
        catch (Exception re)
        {
            throw re;
        }
    }

    public static ArrayList ObtenerTodos(Object example)
    {
        try
        {
            return (ArrayList)NHibernateUtil.getSession()
                .CreateCriteria(example.GetType())
                .List();
        }
        catch (Exception re)
        {
            throw re;
        }
    }
}
```

Figura 2.4 DaoGenerico.

Esto se logra gracias a nhibernate 1.1, que tan solo necesita un conjunto de ficheros de mapeo (hbm.xml) que representen los objetos persistentes y las relaciones que se establecen entre

estos, además de citar con que tablas y columnas de la BD se relacionan las clases y sus propiedades respectivamente ejemplo:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
    <class name="CentroDigitalizacion.Negocio.Entidades.UDocumentales.UnidadesDocumentales,
                CentroDigitalizacion.Negocio.Entidades" table="UNIDADES_DOCUMENTALES"  schema="CENTRO">
        <id name="Id" column="ID" type="Int32">
                <generator class="assigned"/>
        </id>
        <property name="FolioInicio" type="String" column="FOLIO_INICIO" length="30" />
        <property name="FolioFin" type="String" column="FOLIO_FIN" length="30" />
        <property name="NoDocum_nto" type="Int32" column="NO_DOC"/>
        <property name="PaginaInicio" type="Int32" column="PAGINA_INICIO"/>
        <property name="PaginaFin" type="Int32" column="PAGINA_FIN"/>
        <property name="Cancelado" type="Boolean" column="CANCELADO"/>
        <many-to-one name="TomoOb" class="CentroDigitalizacion.Negocio.Entidades.Tomo.Tomo,
                        CentroDigitalizacion.Negocio.Entidades" fetch="select">
            <column name="IDTOMO" />
        </many-to-one>
        <set name="ImagenesUDoc" inverse="true" lazy="true">
            <key>
                <column name="ID_UDOCUMENTALES" not-null="true"/>
            </key>
            <one-to-many class="CentroDigitalizacion.Negocio.Entidades.ImagenUDocumentales.ImagenUDocumentales,
                                CentroDigitalizacion.Negocio.Entidades" />
        </set>
    </class>
</hibernate-mapping>
```

Figura 2.5 XML de mapeo usado por NHibernate.

En este ejemplo se mapea la clase UnidadesDocumentales que se encuentra en el ensamblado CentroDigitalizacion.Negocio.Entidades y se enlaza con su origen de datos en la tabla UNIDADES_DOCUMENTALES de la base de datos. Luego se enlazan todas las propiedades con su correspondiente columna, la propiedad FolioInicio de tipo String se le asocia la columna FOLIO_INICIO y así con todas las propiedades que se desee hacer persistentes. Luego se describe una relación de muchos a uno con la clase Tomo que se encuentra en el ensamblado CentroDigitalizacion.Negocio.Entidades y una relación de uno a muchos con la clase ImagenUDocumentales que pertenece al mismo ensamblado.

Este tipo de mapeo se realiza con cada una de las clases pertenecientes a CentroDigitalizacion.Negocio, que representa las clases entidades.

2.4 Patrón Fachada

Propósito

Proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La "fachada" satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Estructura

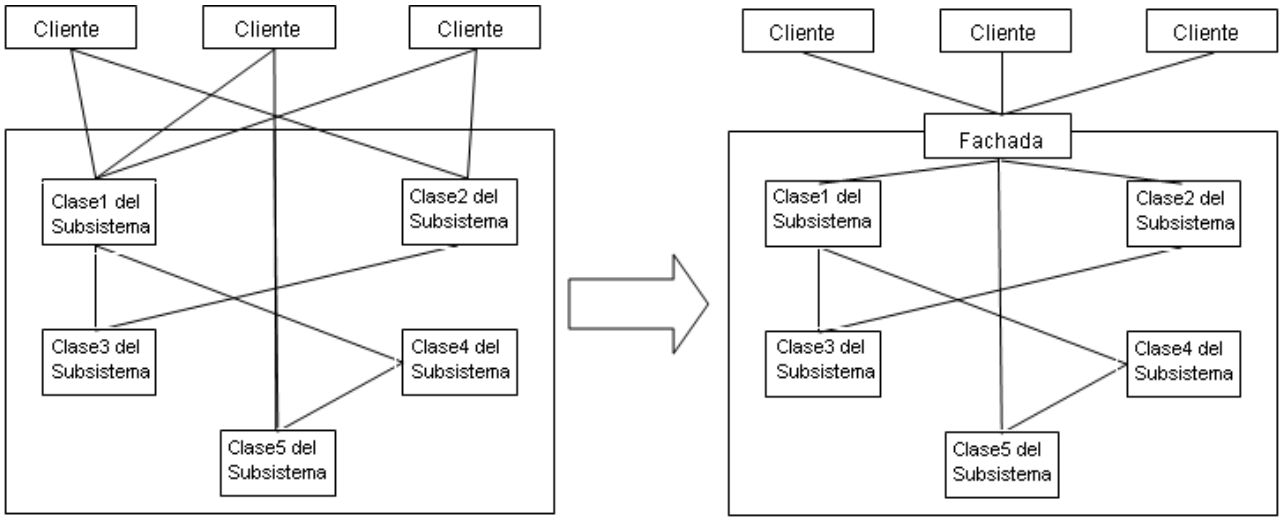

Figura 2.6 Representación del Patrón Fachada.

Uso:

Cuando se pretende proporcionar una interfaz simple para un subsistema complejo.

Cuando existen muchas dependencias entre los clientes y las clases que implementan una abstracción. Una "fachada" proporciona al subsistema independencia y portabilidad.

Cuando se pretende descomponer un sistema en capas el patrón fachada puede ofrecer una interfaz de acceso entre las mismas.

Ventajas:

Al separar al cliente de los componentes del subsistema, se reduce el número de objetos con los que el cliente trata, facilitando así el uso del subsistema.
Oculta a los clientes parte de la complejidad de los subsistemas.

Se promueve un acoplamiento débil entre el subsistema y sus clientes, eliminándose o reduciéndose las dependencias.
No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten. De esta forma podemos elegir entre facilidad de uso y generalidad.

En la solución propuesta disponemos de la librería Nhibernate con un conjunto clases internas que se encargan de la manipulación de datos. En la aplicación definimos la clase **CADatosFachada** la cual implementa el patrón fachada y es la intermediaria entre la capa de acceso a datos y la capa de negocio, esto significa que todas las peticiones de datos irán a parar solo a esta clase.

2.5 Descripción de las clases

Caso de Uso Escanear Tomo

Nombre: accEscanearTomo	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
tomo	Tomo
movimientoFinal	Movimientos
movimientoNota	MovimientoNota
configuracionEscaneo	ConfiguracionScanner
codigoBarra	String
directorioSalva	String

nombreFichero	String
gtrEscanearTomo	GtrEscanearTomo
registro	RegistryKey
listaTomosAtrazados	ArrayList
imagenTomo	ImagenTomo
Para cada operación:	
Nombre:	btnCerrar_Click
Descripción:	Cerrar la interfaz.
Nombre:	btnVistaPrevia_Click
Descripción:	Visualiza en un visor a pantalla completa el tomo actual.
Nombre:	btrNuevo_Click
Descripción:	Salvar el tomo actual si no se ha salvado y limpiar los componentes para escanear uno nuevo.
Nombre:	btnGuardar_Click
Descripción:	Salvar el tomo actual en la BD.
Nombre:	CargarEntidad
Descripción:	Obtiene un tomo y su imagen por el código de barra.
Nombre:	MostrarEntidad
Descripción:	Se encarga de salvar la imagen en disco y visualizarla.
Nombre:	SalvarEntidad
Descripción:	Salvar la imagen en el tomo.
Nombre:	escanearTomo_Load
Descripción:	Inicializa todos los controles visuales.

Nombre: GtrEscanearTomo	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
gtrTomo	GtrTomo
gtrRegMov	GtrRegistrarMovimiento
gtrUsuario	GtrUsuario
gtrConfigGeneral	GtrConfiguracionGeneral

gtrSecuencia	GtrSecuenciaNmMovimiento
Para cada operación:	
Nombre:	ObtenerTomo(código: string, operación: string): Tomo
Descripción:	Obtiene un tomo por el código de barra y la operación.
Nombre:	ObtenerTomo(código: string) : Tomo
Descripción:	Obtiene un tomo por el código de barra.
Nombre:	SalvarTomo(tomo: Tomo)
Descripción:	Salva un tomo.
Nombre:	ConfiguracionScanner() : ObtenerConfiguracionEscaner
Descripción:	Obtener la configuración de la escaner.
Nombre:	ObtenerMovimientoNota(idTomo: int, operación: string): MovimientoNota
Descripción:	Obtiene el movimiento nota asociado al tomo.
Nombre:	SalvarImagenTomo(tomo :Tomo, imagenTomo: ImagenTomo, directorio: string, movimiento: Movimientos)
Descripción:	Carga la imagen del tomo de disco y la salva en la BD.
Nombre:	ObtenerMovimiento(operación: string): Movimientos
Descripción:	Obtiene un movimiento por la operación.
Nombre:	VerificarSecuenciaTomo(idTomo: int, operación: string): bool
Descripción:	Verifica la secuencia del tomo.
Nombre:	ObtenerPorcientoRevision():int
Descripción:	Obtiene el porciento de revisión.
Nombre:	ObtenerImagenTomo(idTomo: int): ImagenTomo
Descripción:	Obtiene la imagen de un tomo.

Caso de Uso Recuperar Unidad Documental

Nombre: accRecuperarUD	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
unidadesD	ArrayList
tomosNoExtisten	int
paginasNoIncl	ArrayList
action	accRecuperando
caminoUnidadesDoc	String []
Para cada operación:	
Nombre:	btnCerrar_Click
Descripción:	Cierra la interaz.
Nombre:	btnExaminar_Click
Descripción:	Permite buscar el camino de las salvas.
Nombre:	btnRecuperar_Click
Descripción:	Recupera los tomos.
Nombre:	btnCargarSalva_Click
Descripción:	Carga todos los tomos a recuperar.

Nombre: gtrRecuperarUD	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
Para cada operación:	
Nombre:	ExisteUDocumental(idTomo: int): bool
Descripción:	Verifica la existencia de una UD.
Nombre:	UnidadesDocTomo(idTomo: int): ArrayList
Descripción:	Obtiene las UD de un tomo.
Nombre:	Salvar(aUD: UnidadesDocumentales, culminar1: bool)
Descripción:	Salva la UD.
Nombre:	SalvarPNI(obj : PaginasNoIncluidas)

Descripción:	Salva las páginas no incluidas.
Nombre:	CargarFicheros(camino: string, formato string): string[]
Descripción:	Devuelve una colección de archivos con la extensión pasada en formato.
Nombre:	DeserializarUD(dir: string): ArrayList
Descripción:	Retorna las unidades documentales del tomo serializado.
Nombre:	ObtenerTomoDeserializarUD(dir: string): Tomo
Descripción:	Retorna el tomo des-serializado
Nombre:	DeserializarPNI(dir: string): ArrayList
Descripción:	Retorna las páginas no incluidas.
Nombre:	EliminarPaginasTomo(idTomo: int)
Descripción:	Elimina las páginas no incluidas del tomo.

Caso de Uso Dividir Unidad Documental

Nombre: accDividirUDoc	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
listaTomo	ArrayList
listaTomoDiv	ArrayList
cantidad	int
Para cada operación:	
Nombre:	Cargar
Descripción:	Obtienes todos los tomos recuperados pero no divididos.
Nombre:	llenarListView
Descripción:	Mostrar los atributos de los tomos en el ListView.
Nombre:	btnTerminar_Click
Descripción:	Cierra la interfaz.
Nombre:	btnDividir_Click
Descripción:	Divide el tomo en UD y se salvarán en el directorio de salva.
Nombre:	gtr_capturarCantidad
Descripción:	Visualiza en la interfaz la cantidad actual de tomos divididos.

Nombre: gtrDividirUDoc	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
codecs	RasterCodecs
informacioImagen	CodecsImageInfo
directorioSalva	string
nombreFichero	string
Para cada operación:	
Nombre:	ObtenerTomos(aOperacionSi: string, aOperacionNo : string): ArrayList
Descripción:	
Nombre:	DividirImagen(aTomos: ArrayList, aOperacion:string) : ArrayList
Descripción:	Divide el tomo en UD y se salvarán en el directorio de salva.
Nombre:	SaveByteToFile(path: String, buffer: byte[])
Descripción:	Salvar el tomo en el directorio de salva.

Caso de Uso Identificar Unidad Documental

Nombre: AccIdentificarUDoc	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
entidad	UnidadDocumental
movimiento	Movimiento
tomo	Tomo
lista	ArrayList<UnidadDocumental>
banderaNuevo	bool
banderaCambio	bool
operacion	string
Para cada operación:	
Nombre:	BtnNuevaEntidad_EventHandlerClick
Descripción:	Método que manipula el Clic del botón Nuevo para crear nueva Unidad Documental
Nombre:	BtnGuardarEntidad_EventHandlerClick

Descripción:	Método que manipula el Clic del botón Guardar la Unidad Documental.
Nombre:	BtnEliminarEntidad_EventHandlerClick
Descripción:	Método que manipula el Clic del botón Eliminar unidad documental.
Nombre:	BtnCerrar_EventHandlerClick
Descripción:	Método que manipula el Click del boton Cerrar
Nombre:	BtnNuevo_EventHandlerClick
Descripción:	Método que manipula el Clic del botón Nuevo
Nombre:	ListViewUnidadDoc_EventHandlerChangue
Descripción:	Método que manipula el listView de las unidades documentales.
Nombre:	CargarEntidad()
Descripción:	Actualiza entidad con los datos de la zona editable y de la banderaNuevo estar activada insertar entidad en la lista
Nombre:	MostrarEntidad()
Descripción:	Método que muestra la entidad en los componentes de la zona editable
Nombre:	MostrarNuevaEntidad()
Descripción:	Método encargado de instanciar una nueva U.Doc en la Entidad y llama a MostrarEntidad().
Nombre:	MostrarTomo()
Descripción:	Método encargado de mostrar Imagen del Tomo y en caso de haber Movimiento mostrar la Nota asociada.
Nombre:	CargarTomo()
Descripción:	Método que le pide al negocio (GtrIdentificarUDoc) un Tomo mediante una operación, o sea le pide al método ObtenerTomo del GtrIdentificarUDoc.
Nombre:	GuardarTomo()
Descripción:	Método que le manda al negocio (GtrIdentificarUDoc) a guardar un Tomo.

Nombre: GtrIdentificarUDoc	
Tipo de clase: Clase Gestora	
Atributo	**Tipo**
Para cada operación:	
Nombre:	GuardarTomo(tomo: Tomo; operacion: string)

Descripción:	Método estático que le solicita al GtrTomo que guarde el tomo y le manda al GtrRegistrarMovimiento a registrar el movimiento final de esa operación.
Nombre:	ObtenerTomo(operacion : string)
Descripción:	Método estático que le solicita al GtrTomo un por una operación y le manda al GtrRegistrarMovimiento a registrar el movimiento inicial de esa operación.

2.6 Tratamiento de errores

Para que la aplicación funcionara adecuadamente frente a cualquier situación que se presentase, ya sea por errores inducidos por el usuario, por el propio sistema o debido a factores externos, fue necesario tener un completo control sobre todos los posible errores a ocurrir en el sistema, de tal forma que si ocurriese alguno se pudiera manejar y tomar las acciones pertinentes para que no comprometan la integridad del sistema ni la integridad de los datos.

La administración de excepciones incluye la detección y generación de excepciones, el diseño de éstas, el flujo de información de las mismas y la publicación de información de las excepciones a los usuarios.

Las excepciones proporcionan un flujo de información ascendente. Esto adquiere particular relevancia cuando se alcanza una interfaz de servicios o de usuario con la que no desea establecer un flujo de la excepción literalmente, sino más bien traducirla a algo que el cliente pueda procesar, y sin exponer información confidencial empresarial o técnica acerca de su aplicación o servicio (como por ejemplo, una cadena de conexión a una base de datos en caso de un error de conexión) que se pueda utilizar contra el sistema o la organización.

Es habitual derivar a dos ramas principales de excepciones: excepciones empresariales y técnicas. Este diseño facilita la detección y publicación del tipo de excepciones.

2.6.1 Administración de excepciones en la interfaz de usuario

Las excepciones son lanzadas en cascada desde la capa de acceso a datos hacia la capa del negocio y finalmente hacia la capa de presentación, donde se mostrarán finalmente y las visualizará:

El código de barra (id: 1000343483) no existe. (Negocio)

El usuario activo fue eliminado. (Negocio)

Se ha producido un fallo en la conexión. (Acceso a datos)

Valor duplicado. (Acceso a datos)

Estas excepciones ya vendrán transformadas de forma que el usuario pueda entenderlas y le facilite la tarea de ubicar el error sin brindar información que ponga en peligro nuestra aplicación.

2.6.2 Administración de excepciones en la capa de negocio

Los componentes del negocio controlan las excepciones que provienen de los componentes de acceso a datos las cuales deberán ser propagadas a las capas superiores.

Por lo general, los componentes del negocio no deberían ocultar ninguna excepción procedente de las capas inferiores. La ocultación de excepciones podría llevar a errores a los procesos del negocio y hacer creer al usuario que determinadas operaciones se han realizado correctamente.

Las excepciones producidas en la capa de negocio son lanzadas como ExepcionNull la cual es una clase que hereda de Exception y que recibe como parámetro el mensaje del error que se pretende manipular.

2.6.3 Administración de excepciones en componentes de acceso a datos

Los componentes de acceso a datos controlan las excepciones derivadas de errores técnicos como:

- Problemas para conectarse con determinado privilegio.
- Que no es posible ejecutar determinado procedimiento almacenado.

Nhibernate por ejemplo, genera varios tipos de excepciones cuando Oracle devuelve un error.

- NHibernate.HibernateException
- NHibernate.ADOException
- NHibernate.TransactionException

La forma más óptima que determinamos para manipular estas excepciones fue con la creación de una clase “UtilExcepcion.cs”, que es la encargada de traducir las excepciones provenientes de la capa de acceso a datos para un mejor entendimiento por parte del usuario.

Si la excepción aun no ha sido tratada por Nhibernate entonces el mensaje todavía mantiene el número de la excepción lanzada por Oracle, el cual es capturado con el objetivo de personalizar el mensaje de error.
Ejemplo:

```
switch(ID)
{
    case "12152":error = "Se ha producido un fallo en la conexión.";break;
    case "12562":error = "Imposible obtener la conexión."; break;
    case "12560":error = "Imposible establecer la conexión."; break;
    case "00001":error = "Valor duplicado."; break;
    ...

}
```

Figura 2.7 Tratamiento de excepciones Oracle.

Otro método usado para impedir posible errores fue mediante validaciones como:
Habilitar y deshabilitar botones según sea requerido.
Ejemplo:

- Solo habilitar eliminar y actualizar si se ha seleccionado un ítem.
- Realizar confirmación antes de cerrar una interfaz.
- Realizar confirmación antes de eliminar un ítem.

2.6.4 Ejemplo de tratamiento de excepciones:

```
private void GuardarDatos_Click(object sender, System.EventArgs e)
{
    try
    {
        GuardarTomo();
    }
    catch(Exception ex)
    {
        unidadDoc.GuardarDatos.Enabled=false;
        MostrarExcepcion(ex);
    }
}
```

Figura 2.8 Tratamiento de excepciones en la capa de interfaz.

```
void GuardarTomo()
{
    if (tomo==null)
        throw new Excepcion("No existe ningún tomo");
    if (¡verificapaginas())
        throw new Excepcion("La cantidad de pág…");
    try
    {
        MostrarReloj();
        GtrIdentificarUDoc gestor=new GtrIdentificarUDoc();
        gestor.GuardarTomo(tomo,"Identificar U.Doc");
        OcultarReloj();
    }
    catch (Exception ee)
    {
        OcultarReloj();
        throw new UtilExcepcion.Excepcion(ee," Tomo ");
    }
}
```

Figura 2.9 Tratamiento de excepciones en la capa de negocio.

2.7 Estándares de Codificación

Las convenciones de codificación son importantes y los desarrolladores por varias razones:

- Mejoran la legibilidad de los artefactos software.
- Reducen el tiempo y el esfuerzo del entrenamiento

2.7.1 Organización de los ficheros

2.7.1.1 Ficheros de código

Clases/ficheros cortos, que no excedan las 2000 líneas de código.

Hacer las estructuras/clases claras.

Cada clase en un fichero independiente, con nombre igual que la clase.

2.7.1.2 Directorio

Crear un directorio para cada espacio de nombre. Por ejemplo, para **MiProyecto.Util.Matematica** usar **MiProyecto/Util/Matematica**, no usar puntos en el nombre de los espacios de nombre.

2.7.2 Líneas en blanco

Las líneas en blanco mejoran la legibilidad. Una línea en blanco debe ser usada entre:

- Métodos.
- Propiedades.
- Variables locales y la primera declaración.
- Secciones lógicas dentro de un método.

2.7.3 Número de declaraciones por línea

Una declaración por línea es recomendada, ejemplo:

- int nivel; // nivel de indentación
- int tamaño; // tamaño de la tabla

No poner más de una variable o variables de diferentes tipos en una misma línea; tener en cuenta que los nombres deben ser lo más declarativo posible.

2.7.4 Estilos de capitalización

2.7.4.1 Estilo de Pascal

Capitaliza la primera letra de cada palabra, ejemplo: **UnContador**.

2.7.4.2 Estilo camello

Capitaliza la primera letra de cada palabra, excepto para la primera palabra, ejemplo **unContador**.

2.7.5 Directivas de declaración

El uso de underscore es considerado una mala práctica. Fue utilizado un tiempo atrás para describir el tipo de una variable, pero ya debe considerarse obsoleto.

Un buen nombre de variable describe la semántica, no el tipo.

Con la excepción del código relacionado con la interfaz gráfica de la aplicación. Todos los nombres de variables y campos que contengan elementos de interfaz como botones, se les debe agregar al principio la abreviatura del tipo. Por ejemplo:

System.Windows.Forms.Button **btnCancel**;

System.Windows.Forms.TextBox **txtName**;

2.7.5.1 Abreviaturas más utilizadas

Tipo	**Abreviatura**
System.Windows.Forms.Button	**btn**
System.Windows.Forms.TextBox	**txt**
System.Windows.Forms.ComboBox	**cbx**
System.Windows.Forms.ListBox	**lbx**

System.Windows.Forms.ListView	**lvw**
System.Windows.Forms.DateTimePicker	**dtp**

2.7.6 Clases

Los nombres de las clases deben ser sustantivos o frases en sustantivo.

Usar el estilo de Pascal.

No usar ningún prefijo.

Las clases que heredan de Form, se les agrega el prefijo **frm**.

Las clases que heredan de Accion y AccionSegura, se les agrega el prefijo **acc**.

Las clases gestoras, se les agrega el prefijo **gtr**.

2.7.7 Interfaces

Utilizar sustantivos, frases en sustantivo o adjetivos que describan comportamiento.

Usar el estilo de Pascal.

Usar I como prefijo, seguido por una letra mayúscula (primera letra del nombre de la interfaz).

2.7.8 Enumerados

Usar el estilo de Pascal.

No poner prefijos (sufijos) al identificador ni a los valores.

Identificadores en singular.

2.7.9 Campos de solo lectura y constantes

Utilizar sustantivos o frases en sustantivo.

Usar el estilo de Pascal.

2.7.10 Parámetros/campos no constantes

Usar nombres descriptivos, deben ser suficientes para determinar el significado de la variable y su tipo, preferiblemente su significado.

Usar el estilo camello.

2.7.11 Variables

Utilizar i, j, k, l, m, n para contadores en ciclos triviales, en caso contrario utilizar nombres de variables más descriptivos.

Utilizar el estilo camello.

2.7.12 Métodos

Nombrar los métodos con verbos o frases verbales.

Utilizar el estilo de Pascal.

2.7.13 Propiedades

Nombrar las propiedades utilizando sustantivos o frases en sustantivo.

Utilizar el estilo de Pascal.

Considerar nombrar las propiedades con el mismo nombre de su tipo.

2.7.14 Eventos

Nombrar los manejadores de eventos con el sufijo EventHandler.

Usar dos parámetros nombrados sender y e.

Usar el estilo de Pascal.

Nombrar las clases que sean para pasar argumentos con el sufijo EventArgs.

Nombrar los eventos que utilizan el concepto de presente y pasado utilizando el tiempo en que ocurren.

Considerar nombrar utilizando verbos.

2.8 Conclusiones

En este capítulo hemos concluido que el uso de patrones de arquitectura unido a los estándares de codificación, el tratamiento de errores utilizados, ha facilitado el desarrollo del sistema disminuyendo el tiempo de desarrollo y aumentando la claridad del sistema, para un fácil entendimiento de las líneas de código.

Capitulo 3. Validación de la solución propuesta

3.1 Introducción

La aplicación a lo largo de su vida es sometida a una serie de pruebas, tanto por los programadores, jefes de equipo, y el personal de calidad. Cada vez que se realizan cambios, nuevas funcionalidades, se realizan nuevas pruebas para examinar el correcto funcionamiento del sistema.

Las pruebas a realizar en tiempo de desarrollo son aquellas que hace el desarrollador en su oficina, tienen como objetivo comprobar que el programa compile y ver que todo está yendo como debiera, normalmente se realizan varios cientos de estas pruebas que básicamente consisten en compilar periódicamente durante el desarrollo y ejecutar para ver el resultado.

Dentro de las pruebas en tiempo de desarrollo encontraremos las pruebas unitarias, estas son pruebas de menor escala y consisten en probar cada uno de los módulos que conforman el programa. Cuando estos módulos son extensos o complejos se dividen para probar objetivamente partes más pequeñas, este tipo de pruebas es la más común en esta fase.

Los test de unidad nos ayudan a comprobar el correcto comportamiento de nuestro código así como a detectar fallos tanto de comportamiento como de concepto y contribuye a desarrollar un software de mayor calidad.

3.2 Pruebas de unidad

Es típico que los programadores verifiquen un algoritmo a través del depurador del entorno de desarrollo. Ejecutando paso a paso podemos ir comprobando el valor de las variables, y verificando así que cada variable toma el valor adecuado en cada momento. Este es un proceso lento y complicado, ya que requiere mucha atención del programador, para ir comprobando todas

las variables, y es muy fácil perderse en algún detalle o equivocarse en alguna operación. Además, el proceso de depuración requiere mantener la atención al máximo durante mucho tiempo, y muchas personas no son capaces de mantener ese nivel de concentración durante un proceso de depuración largo, de por ejemplo, una o dos horas.

La filosofía de las pruebas unitarias es la de verificar aisladamente cada método implementado en un módulo o clase. Sin embargo, no se puede olvidar que dichas clases conforman un sistema estrechamente interrelacionado, de manera que entre una y otra puede haber relaciones de colaboración y/o dependencia. Para lograr el aislamiento, se debe recurrir a la construcción de un software adicional que genere las condiciones de trabajo requeridas por los métodos de la clase.

Una prueba de unidad debe funcionar extremadamente rápido. Si necesita esperar conexiones de la base de datos o procesos externos del servidor, o analizar archivos grandes, su utilidad rápidamente va a ser limitada. Una prueba debe proporcionar una respuesta inmediata y una satisfacción instantánea.

3.2.1 Pautas para el desarrollo de las pruebas de unidad:

Probamos los métodos con los valores que no esperamos recibir. Para comenzar, probamos los métodos con ceros, valores nulos y las condiciones que son definidas por la lógica del negocio como máximo o mínimo en los valores de la entrada. Crear las pruebas para todos estos casos extremos.

Una vez que tengamos las pruebas para los casos extremos, escribir simplemente las pruebas para los casos comunes. Para estas pruebas utilizamos los valores de las condiciones que ocurrirán más frecuentemente. Los casos extremos son ciertamente más importantes de probar, porque ellos son olvidados o descuidados por el programador, pero los casos comunes deben siempre estar probados.

Con los casos comunes se le da una ojeada al algoritmo que se está probando e identificar todas las ramas y puntos de decisión. Para tener un sistema completo de pruebas, cada rama y la

condición deben ser probadas. Esto también se aplica dondequiera que exista excepción a ser lanzada.

Casos extremos: son las condiciones que pudieron exponer un problema porque están cerca o en del borde de valores aceptables.

Ventajas:

- **Los errores son más fáciles de localizar.**
- **Se reducen los "efectos secundarios":** muchas veces, cuando queremos arreglar algo bajo presión, cometemos otros errores, o no tenemos en cuenta ciertos aspectos, que hacen que el programa deje de funcionar por otro sitio. Incluso a veces, es más peligroso arreglar un error que dejarlo como está, ya que podemos subsanar el error, pero generar otros distintos.
- **Se da más seguridad al programador:** Las pruebas unitarias aseguran que una corrección no repercute en otros módulos, y permite al programador centrarse en la corrección del error, y no en la repercusión que puede tener esa corrección.
- **Los errores se detectan antes que de otra forma:** De hecho, con pruebas unitarias, la mayoría de los errores de programación se detectan durante la propia etapa de programación.
- **Cada prueba se convierte en un ejemplo de uso:** Con el conjunto de pruebas, ponemos a disposición un conjunto bastante amplio de ejemplos.

A la solución propuesta se le hizo pruebas unitarias solo a la fachada por ser la que controla todo el flujo de información proveniente de o hacia la base de datos. Para esto se creó un proyecto denominado CentroDigitalizacion.Test con el objetivo de que no interfiera en el código de la aplicación y en él se realizan todas las pruebas necesarias. Inicialmente se hicimos las pruebas de las operaciones CRUD (create, retrieve, update, delete) de la fachada.

- Para ello, se crearon los metodos **TestSalvarSeleccionar**, **TestBorrar**, **TestActualizar** como se muestra a continuación.

```
public class CADatosFachadaTest
{
    public CADatosFachadaTest()
    {
    }

    Persona persona1;
    Persona persona2;

    [SetUp]
    protected void SetUp()
    {
        //se inicializan dos objetos personas
        persona1 = new Persona();
        persona1.Nombre = "Junior";
        persona1.Direccion = "Calle Cualquiera";
        persona1.CodigoBarra = "10000000";
        persona1.Activo = true;
        persona1.Cedula = "12365478965";
        persona2 = new Persona();
        persona2.Nombre = "Rene";
        persona2.Direccion = "Calle Cualquiera";
        persona2.CodigoBarra = "10100000";
        persona2.Activo = true;
        persona2.Cedula = "24566789907";
    }
```

Figura 3.1 Clase diseñada para realizar las pruebas de unidad.

```
public void TestSalvar ()
{
    /*Es un salvar para cualquier tipo de clase
        del negocio en este caso escogimos a persona*/
    CADatos.CADatosFachada.SalvarNuevoE(persona1);
    /*Comprobar que el id generado por la base de
        datos no sea vacio */
    Assert.IsNotNull(persona1.Id);
    /*Buscar la persona con ese id y comprobar que
        es la misma que insertamos*/
    Persona persona = (Persona)
        CADatos.CADatosFachada.BuscarPorId
        (new Persona(),persona1.Id);
    Assert.AreEqual("12365478965",persona.Cedula);
}

[Test]
public void TestBorrar()
{
    CADatos.CADatosFachada.Salvar(persona2);
    Assert.IsNotNull(persona2.Id);
    CADatos.CADatosFachada.Eliminar(persona2);
    Persona persona = (Persona)
        CADatos.CADatosFachada.BuscarPorId
        (new Persona(),persona2.Id);
    Assert.IsNull(persona);
}
```

Figura 3.2 Clase diseñada para realizar las pruebas de unidad (Continuación).

```
[Test]
public void TestActualizar()
{
    /*Es un salvar para cualquier tipo de clase del
        negocio en este caso escogimos a persona*/
    CADatos.CADatosFachada.Salvar(persona1);
    /*comprobar que el id generado por la base de
        datos no sea vacio */
    Assert.IsNotNull(persona1.Id);
    /*buscar la persona con ese id y comprobar que
        es la misma que insertamos*/
    Persona persona = (Persona)
        CADatos.CADatosFachada.BuscarPorId
        (new Persona(),persona1.Id);
    //verificar que su cedula es la  correcta
    Assert.AreEqual("12365478965",persona.Cedula);
    //Cambiar el nombre
    persona1.Nombre="Alain";
    //actualizar
    CADatos.CADatosFachada.Salvar(persona1);
    //volver a seleccionarlo de la BD
    Persona = (Persona)CADatos.CADatosFachada.BuscarPorId
        (new Persona(),persona1.Id);
    //Comprobar que se ha realizado la actualizacion
    Assert.AreEqual("Alain",persona.Nombre);
}
```

Figura 3.3 Clase diseñada para realizar las pruebas de unidad (Continuación).

```
    [Test]
    public void TestSeleccion()
    {
        IList lista = CADatos.CADatosFachada.ObtenerTodos
            (new Persona());
        Persona p1 = new Persona();
        p1.Id=1;
        p1.Nombre = "Pepe";
        p1.Direccion = "Granma";
        p1.CodigoBarra = "10000300";
        p1.Activo = true;
        p1.Cedula = "89365478965";
        Persona p2 = new Persona();
        p1.Id=2;
        p2.Nombre = "Paco";
        p2.Direccion = "Holguin";
        p2.CodigoBarra = "10102000";
        p2.Activo = true;
        p2.Cedula = "56566789907";
        CollectionAssert.IsSubsetOf(new Persona[]{p1,p2},lista);
    }
    //....
}
```

Figura 3.4 Clase diseñada para realizar las pruebas de unidad (Continuación).

En el método **TestSalvarSeleccionar** se crea una persona y luego la recupera de la base de datos y verifica que sea la misma que se salvó.

En el método **TestBorrar** se crea una persona y se salva en la base de datos y luego se borra usando su identificador.

El método **TestActualizar** se crea una persona y luego se selecciona y se le actualizan los datos.

Ya en el caso del método **TestSeleccion**, se llena la base de datos con datos para prueba de forma manual, luego se seleccionan todas las personas de la Base de Datos y se comprueba que

los datos seleccionados para la prueba existan en la lista de personas retornada de la base de datos.

3.3 Pruebas de caja negra

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra pretende demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta, y
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.
La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

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

Algunas clases de pruebas:

- Cubrimiento (Invocar todas las funciones).
- Pruebas de valores límite (Probar la validación de los rangos válidos).

3.4 Casos de prueba

3.5 CPR 1: Devolver Tomos

Descripción General

Durante el desarrollo del caso de uso devolver tomo no fue detectada ninguna incidencia a tener en cuenta para el diseño de las pruebas.

A este Caso de Uso se le realizaron las siguientes pruebas:

- Prueba sobre el TextBox del código de barra.

Descripción de la Funcionalidad:

El funcionario entra el código de barra del transportista que va a recoger los tomos, si el código de barra es incorrecto el sistema muestra que el transportista no está registrado, sino, habilita el ComboBox lote de salida, donde el funcionario selecciona el lote que se va a llevar el transportista, el sistema muestra la cantidad de lotes existentes en ese lote en el TextBox cantidad de tomos y en el ListView visualiza los tomos, para finalizar la operación el funcionario pulsa el botón guardar, quedando registrado en el sistema que el transportista recogió el lote en cuestión para ser devuelto a los Registros y Notarías del MIJ de Venezuela.

Flujo Central:

- El funcionario da click en la opción del menú “Proceso de Digitalización”.
- El funcionario da click en la opción del submenú “Devolver Tomo”.
- El funcionario lee el código de barra del transportista.
- El funcionario selecciona el lote de salida.
- El funcionario da click en guardar y finaliza el caso de uso.

Condiciones de Ejecución:

El código de barra del transportista debe ser válido.

Iteraciones

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El recepcionista captura un código de barra válido, CB: TT22222222		El sistema muestra el nombre del transportista.	El sistema mostró el nombre del transportista.	
	El recepcionista captura un código de barra válido, CB: TT22222225	El sistema muestra que el código de barra no está registrado en el sistema.	El sistema mostró que el código de barra no está registrado en el sistema.	

3.6 CPR 2: Exportar Tomos

Descripción General

Durante el desarrollo del caso de uso exportar tomo no fue detectada ninguna incidencia a tener en cuenta para el diseño de las pruebas.

A este Caso de Uso se le realizaron las siguientes pruebas:

- Prueba sobre los RadioButton tomos a exportar.
- Prueba sobre los RadioButton cantidad a exportar.
- Pruebas sobre el TextBox otra cantidad.

Descripción de la Funcionalidad:

El funcionario selecciona una opción de los tomos a exportar, todos o sin exportar, la opción todos como su nombre lo indica, muestra todos los tomos, hayan sido o no exportados, y la opción sin exportar solo muestra aquellos que no han sido exportados, una vez seleccionada

una opción se habilita el botón cargar tomos, una vez pulsado el click sobre este botón se mostraran los tomos, luego se selecciona la opción de la cantidad a exportar, todos u otra cantidad, si selecciona todos se exportaran todos, sino se exportará solo la cantidad especificada, en el orden en que se muestran, luego se pulsa el botón exportar tomos y los tomos son exportados.

Flujo Central:

- El funcionario da click en la opción del menú “Unidades Documentales”.
- El funcionario da click en la opción del submenú “Exportar Tomo”.
- El funcionario selecciona los tomos a exportar, todos o sin exportar.
- El funcionario da click en cargar tomos.
- El funcionario selecciona la cantidad de tomos a exportar, todos u otra cantidad.
- El funcionario da click en exportar tomos.

Condiciones de Ejecución:

La cantidad de tomo a exportar debe ser menor o igual a la cantidad mostrada.

Iteraciones

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
La cantidad de tomo es menor o igual a la cantidad mostrada, cantidad mostrada: 5 cantidad a export: 3		El sistema exporta correctamente la cantidad de tomos seleccionada.	El sistema exportó correctamente los tomos.	
	La cantidad de tomo es menor o igual a la cantidad mostrada, cantidad mostrada: 5 cantidad a export: 7	El sistema muestra una advertencia, especificando que la cantidad establecida excede a la cantidad disponible.	El sistema mostró una advertencia especificando que la cantidad establecida excede a la cantidad disponible.	

3.7 CPR 3: Datos de Tomo

Descripción General

Durante el desarrollo del caso de uso datos de tomo no fue detectada ninguna incidencia a tener en cuenta para el diseño de las pruebas.

A este Caso de Uso se le realizaron las siguientes pruebas:

- Pruebas sobre el TextBox código de barra.

Descripción de la Funcionalidad:

El funcionario captura el código de barra del tomo, el sistema muestra los datos del mismo en caso de ser un tomo válido, de lo contrario advierte que el tomo no existe, si el tomo es válido permitirá al funcionario eliminar su último movimiento.

Flujo Central:

- El funcionario da click en la opción del menú “Proceso de Digitalización Documentales”.
- El funcionario da click en la opción del submenú “Mostrar Datos Tomo”.
- El funcionario captura el código de barra del tomo.
- El funcionario elimina el último movimiento del tomo mostrado.

Condiciones de Ejecución:

El código de barra debe ser válido.

Iteraciones

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El recepcionista captura un código de barra válido, CB: TM00000000		El sistema muestra los datos del tomo.	El sistema mostró los datos del tomo.	
	El recepcionista captura un código de barra válido, CB: TM00000001	El sistema muestra una advertencia, el tomo no existe.	El sistema mostró una advertencia, el tomo no existe.	

3.8 Pruebas de caja blanca

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que:

- Se ejercitan todos los caminos independientes de cada módulo.
- Se ejercitan todas las decisiones lógicas.
- Se ejecutan todos los bucles.
- Se ejecutan las estructuras de datos internas.

Algunas clases de pruebas:

- *Pruebas de cubrimiento* (Ejecutar al menos una vez cada sentencia).
- *Pruebas de condiciones* (Cada condición debe cumplirse en un caso y en otro no).
- *Pruebas de bucles*.

3.9 Pruebas de regresión

Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad.

El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido.
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior.

3.10 Complejidad ciclomática

Es una métrica que mide el número de decisiones lógicas en un segmento de código. Posee además dos componentes, descriptivo y prescriptivo.

Descriptivo, es aquel que identifica código susceptible a errores, difícil de entender, difícil de modificar, difícil de probar y demás.

Prescriptivo, es aquel que identifica los pasos operacionales para ayudar a controlar el software, por ejemplo: dividir módulos altamente complejos en varios módulos de menor complejidad o indicar la cantidad de pruebas que deben ejecutarse para cada módulo.
Refleja una medida de la complejidad del código.

Existe una fuerte conexión entre la complejidad ciclomática y las pruebas de software:

Primero, la complejidad es una fuente común de errores en el software. Esto se manifiesta de forma abstracta y concreta:

- **Abstracta**, la complejidad más allá de ser una medida, es una resultante de la capacidad humana de realizar operaciones exactas.
- **Concreta**, numerosos estudios y experiencia de la industria han demostrado que el grado de complejidad tiene una correlación con los errores en el software.

Por lo anterior muchas organizaciones han decidido medir la complejidad ciclomática de sus desarrollos y limitarla con la finalidad de incrementar los niveles de certeza de su software.

Segundo, la complejidad puede ser utilizada para definir el esfuerzo de pruebas, esto se logra haciendo énfasis en aquellos elementos de software de alta complejidad, los cuales son elementos con mayor susceptibilidad a error. (11)

La complejidad de McCABE V (G) (complejidad ciclomática):

- La métrica de McCabe ha sido muy popular en el diseño de pruebas.
- Es un indicador del número de caminos independientes que existen en un grafo. (12)

La complejidad de McCabe se puede calcular de cualquiera de estas 3 formas

1. V (G) = a - n + 2, siendo a el número de arcos o aristas del grafo y n el número de nodos.
2. V (G) = r, siendo r el número de regiones cerradas del grafo.
3. V (G) = c + 1, siendo c el número de nodos de condición.

Figura 3.5 Formulas para calcular V(G).

En este ejemplo podemos apreciar cómo se calcula:

```
5  private double Promedio(Estudiante estudiante)
   {
       int cant5 = 0;                                  //1
       int cant = 0;                                   //1
       int suma = 0;                                   //1
       double promedio = 0;                            //1
       int i = 0;                                      //1
       while (i < estudiante.nota.length)              //2
       {
           int nota = estudiante.nota[0];              //3
           if ((nota >= 2) && (nota <= 5))             //4 y 5
           {
               if (nota == 5)                          //6
                   cant5++;                            //7
               else
                   cant++;                             //8
                suma += nota;                          //9
           }
           i++;                                        //10
       }
       promedio = suma / (cant + cant5);               //11
       return promedio;                                //11
   }
```

Figura 3.6 Segmento de código de ejemplo.

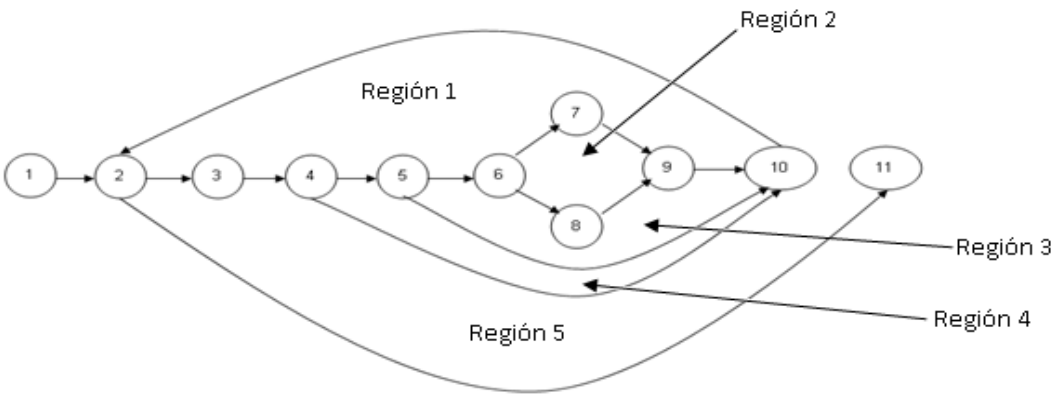

Figura 3.7 Grafo de flujo del segmento de código.

V (G) = a - n + 2 = 14 – 11 +2 = 5.

V (G) = r = 5.

V (G) = c + 1 = 4 +1 = 5.

Como podrán observar por cualquiera de las 3 vías anteriores obtendremos que V (G) = 5.

En las siguientes figuras se pueden apreciar la complejidad ciclomática de los métodos del sistema (el número azul a la izquierda de cada método), que en la mayoría de los casos es relativamente baja, con valores entre 1 y 3, en algunas situaciones alcanza valores superiores, y en situaciones poco frecuente alcanza valores entre 16 y 19.

```
1  public accRecuperarUD(): base("Recuperar U. Documentales","Recuperar Unidades Documentales")...
1  protected override System.Windows.Forms.Form CrearForma()...
1  private void btnCerrar_Click(object sender, EventArgs e)...
7  private void btnExaminar_Click(object sender, EventArgs e)...
3  private void btnRecuperar_Click(object sender, EventArgs e)...
1  private void form_Load(object sender, EventArgs e)...
2  private void acc_capturarTomos(object sender, Negocio.Entidades.Tomo.Tomo aTomo)...
1  private void acc_cerrarForma(object sender, object accion, bool cerrada)...
7  private void btnCargarSalva_Click(object sender, EventArgs e)
   {
       frmRecuperarUD form = (frmRecuperarUD)this.Formulario;
       try
       {
           MostrarReloj();
           form.listRecuperados.Items.Clear();
           form.listConErrores.Items.Clear();
           tomosNoExtisten = 0;
           unidadesD = new ArrayList();
           Negocio.RecuperarUD.gtrRecuperarUD gtrRecuperar = new gtrRecuperarUD();
           Negocio.Entidades.Tomo.Tomo tomo = new CentroDigitalizacion.Negocio.Entidades.Tomo.Tomo();
           ArrayList uDocumentales = new ArrayList();
           ArrayList paginasNoInc = new ArrayList();
           caminoUnidadesDoc = gtrRecuperar.CargarFicheros(form.textCaminoSalvas.Text, "*.txt");
           ListViewItem item = new ListViewItem();
```

Figura 3.8 Ejemplos de complejidad ciclomática.

```
            //Limpiar para que no se carge varias veces las de un mismo tomo
            paginasNoIncl.Clear();
            unidadesD.Clear();
            for(int i = 0; i < caminoUnidadesDoc.Length; i++)
            {
                tomo = gtrRecuperar.ObtenerTomoDeserializarUD((string)caminoUnidadesDoc[i]);
                if(tomo.Id != 0 || tomo != null)
                {
                    item = form.listRecuperados.Items.Add(tomo.ProtocoloOb.Nombre);
                    item.SubItems.Add(tomo.OficinaOb.Nombre);
                    item.SubItems.Add(tomo.Ano.ToString());
                    item.SubItems.Add(tomo.Trimestre.ToString());
                    item.SubItems.Add(tomo.Numero.ToString());
                }
                else
                    tomosNoExtisten++;
            }
            form.btnRecuperar.Enabled = true;
            form.btnCargarSalva.Enabled = false;

            if(caminoUnidadesDoc.Length == 0)
            {
                frmMensaje mensaje = new frmMensaje();
                mensaje.Encabezado = "Información";
                mensaje.Mensaje = "El camino seleccionado no contiene salvas.";
                mensaje.Show();
            }
            OcultarReloj();
```

Figura 3.9 Ejemplos de complejidad ciclomática (Continuación).

```
        }
        catch(Exception ex)
        {
            OcultarReloj();
            if(form.listRecuperados.Items.Count > 0)
            {
                form.btnRecuperar.Enabled = true;
                form.btnCargarSalva.Enabled = false;
            }
            MostrarExcepcion(ex);
        }
    }

}
```

Figura 3.10 Ejemplos de complejidad ciclomática (Continuación).

```
1   public accDividirUDoc(): base("Dividir en U. Documentales","Dividir en unidades documentales")...
1   protected override System.Windows.Forms.Form CrearForma()...
2   public void Cargar()...
2   public void llenarListView()...
2   private void form_Load(object sender, EventArgs e)...
1   private void btnTerminar_Click(object sender, EventArgs e)...
8   private void btnDividir_Click(object sender, EventArgs e)...
3   private void textBoxEsp_KeyPress(object sender, KeyPressEventArgs e)...
1   private void radioButtonTodo_Click(object sender, EventArgs e)...
1   private void radioButtonEsp_Click(object sender, EventArgs e)...
1   private void gtr_capturarCantidad(object sender)...
```

```
1   protected override System.Windows.Forms.Form CrearForma()...
1   private void form_Load(object sender, EventArgs e)...
5   private void btnCargar_Click(object sender, EventArgs e)...
8   private void btnExportar_Click(object sender, EventArgs e)...
1   private void radioBtnTodos_Click(object sender, EventArgs e)...
1   private void radioBtnSinExportar_Click(object sender, EventArgs e)...
5   private void textBoxOtraCant_KeyPress(object sender, KeyPressEventArgs e)...
    /**/
1   private void radioBtnCanTotal_Click(object sender, EventArgs e)...
1   private void radioBtnOtraCant_Click(object sender, EventArgs e)...
1   private void textBoxOtraCant_TextChanged(object sender, EventArgs e)...
1   private void btnCerrar_Click(object sender, EventArgs e)...
```

```
1   public accRecupTomos() :  base("Recuperación de Tomos","Recuperación de Tomos")...
    public GtrRecuperarTomo gtrRecTomos=null;
1   protected override System.Windows.Forms.Form CrearForma()...
2   void CargarTransportista()...
2   private void tomos_Load(object sender, EventArgs e)...
3   private void btnExaminar_Click(object sender, EventArgs e)...
1   private void btnCerrar_Click(object sender, EventArgs e)...
1   private void btnCancelar_Click(object sender, EventArgs e)...
6   private void btnCargar_Click(object sender, EventArgs e)...
2   private void btnGuardar_Click(object sender, EventArgs e)...
1   private void comboBoxTransp_SelectedIndexChanged(object sender, EventArgs e)...
1   private void textCamino_TextChanged(object sender, EventArgs e)...
1   private void gtrRecTomos_Porciento(int porciento)...
```

```
1   public accRecepcionDVD() : base("Recepción DVD","Recepción DVD")...
1   protected override System.Windows.Forms.Form CrearForma()...
2   private void forma_Load(object sender, EventArgs e)...
3   private void txtCantidad_KeyPress(object sender, KeyPressEventArgs e)...
1   private void LimpiarTxt()...
1   private void HabilitarBtn(bool nuevo,bool editar,bool guardar,bool eliminar,bool salvar)...
1   private void HabilitarTxt(bool id,bool cant,bool tx,bool est)...
1   private void btnNuevo_Click(object sender, EventArgs e)...
6   private bool Validar()...
4   private void btnGuardar_Click(object sender, EventArgs e)...
1   private void lstDVD_Click(object sender, EventArgs e)...
1   private void btnEditar_Click(object sender, EventArgs e)...
1   private void btnEliminar_Click(object sender, EventArgs e)...
3   private void btnCerrar_Click(object sender, EventArgs e)...
2   private void btnSalvar_Click(object sender, EventArgs e)...
6   private void lstDVD_KeyDown(object sender, KeyEventArgs e)...
```

Figura 3.11 Ejemplos de complejidad ciclomática.

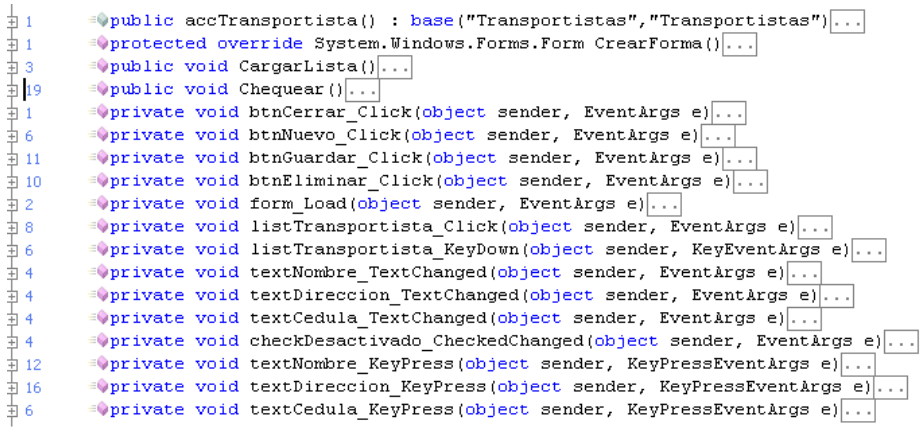

Figura 3.12 Ejemplos de complejidad ciclomática.

Como conclusión podemos decir que los códigos del sistema no son complejos en general, son legibles, y el esfuerzo requerido durante las pruebas será muy bajo.

3.11 Recomendaciones

Como recomendaciones, para los nuevos módulos a realizar en el futuro contamos con una serie de componentes visuales y no visuales que nos facilitarán la programación, realizando todas las restricciones, validaciones en el tiempo de diseño de la aplicación; en el tiempo de ejecución estas validaciones se verificarán de forma autónoma por los componentes para que sean cumplidas, lo que facilita al programador la posibilidad de solo dedicarse a la programación de las funcionalidades del caso de uso, olvidándose por completo de las validaciones de los datos entrados por el usuario.

Entre los componentes mencionados, actualmente funcionales, podemos encontrar:

1. **Visuales:**
   - **CTextBox:** Encargado de la entrada de datos alfanuméricos y numéricos, para entrar cadenas (string), números (int16, int32, int 64, double, decimal y monedas).
   - **CCheckBox:** Encargado de la entrada de parámetros booleanos, verdadero o falso.
   - **CListView:** Encargado de visualizar y manipular colecciones de objetos.

2. **No Visuales:**
    - **CManagerTextCheckBox:** Encargado de manipular todos los controles (CTextBox y CCheckBox) asociados al mismo como un todo.
    - **CCondicion:** Encargado de crear las validaciones y determinar que sean cumplidas.

Además de los componentes antes mencionados se encuentran en planes de desarrollo otros más, CRadioButton, CComboBox, CDateTimePicker, con el objetivo de facilitar la programación, disminuir la complejidad de los algoritmos y disminuir el tiempo de desarrollo del sistema.

Ventajas

- Fácil acceso a la captura de los datos entrados por el usuario y para la visualización de los mismos.
- Las validaciones y restricciones se realizan en el tiempo de diseño.
- Una reducción cuantiosa del número de líneas de código.
- Legibilidad en el código.

En la siguiente figura podemos observar un conjunto de propiedades nuevas de los componentes que dan soporte a las nuevas funcionalidades:

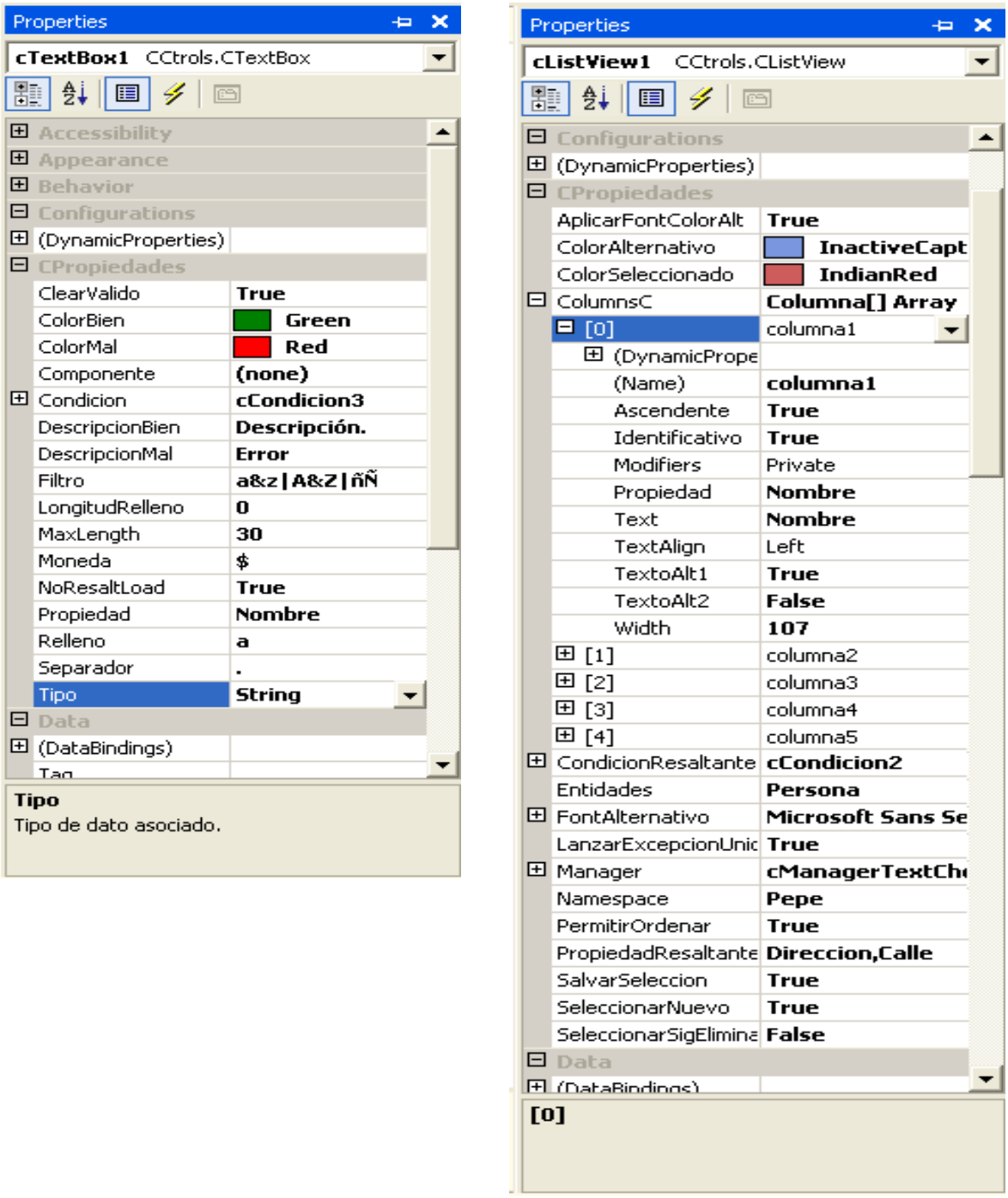

Figura 3.13 Propiedades nuevas de los componentes desarrollados.

En las siguientes figuras podemos observar una cuantiosa diferencia de número de líneas de códigos usando los componentes y sin usar los componentes:

```
        cargar la lista de Protocolo.
public void CargarLista()
{
    frmProtocolo form = (frmProtocolo)this.Formulario;
    GtrProtocolo gtrProtocolo = new GtrProtocolo();
    ArrayList array = gtrProtocolo.ObtenerTodos();
    string nombre = "";
    string codigo = "";
    int cantidad = array.Count;
    for(int i = 0; i < cantidad; i++)
    {
        nombre=((Protocolo)array[i]).Nombre;
        codigo=((Protocolo)array[i]).Codigo;
        form.listProtocolo.Items.Add(Convert.ToString(i+1));
        form.listProtocolo.Items[form.listProtocolo.Items.Count-1]
            .Tag = ((Protocolo)array[i]);
        form.listProtocolo.Items[form.listProtocolo.Items.Count-1]
            .SubItems.Add(codigo);
        form.listProtocolo.Items[form.listProtocolo.Items.Count-1]
            .SubItems.Add(nombre);
    }
}
```

Figura 3.14 Ejemplo de segmento de código sin usar los componentes (21 líneas de código).

```
        cargar la lista de Protocolo.
public void CargarLista()
{
    frmProtocolo form = (frmProtocolo)this.Formulario;
    GtrProtocolo gtrProtocolo = new GtrProtocolo();
    form.listProtocolo.Objetos = gtrProtocolo.ObtenerTodos();
}
```

Figura 3.15 Ejemplo de segmento de código usando los componentes (6 líneas de código).

```
btnEliminar_Click(object sender, EventArgs e), boton eliminar
private void btnEliminar_Click(object sender, EventArgs e)
{
    frmProtocolo form = (frmProtocolo)Formulario;
    if(form.listProtocolo.SelectedIndices.Count != 0)
    {
        int cantidad;
        frmConfirmar confirmar = new frmConfirmar();
        confirmar.Mensaje = "Está seguro que desea eliminar este elemento";
        confirmar.Encabezado = "Alerta";
        DialogResult verificar = confirmar.ShowDialog();
        if(verificar == DialogResult.OK)
        {
            GtrProtocolo gtrProtocolo = new GtrProtocolo();
            int j = form.listProtocolo.SelectedIndices[0];
            try
            {
                gtrProtocolo.Eliminar((form.listProtocolo.Items[j].Tag
                    as Protocolo));
                form.listProtocolo.Items.RemoveAt(j);
                cantidad = form.listProtocolo.Items.Count;
                for(int i = j; i < cantidad; i++)
                {
                    form.listProtocolo.Items[i].Text = (i + 1).ToString();
                }
                if(j < cantidad)
                {
                    form.listProtocolo.Items[j].Selected = true;
                    index = j;
                }
                else if(cantidad > 0 && j == cantidad)
                {
                    form.listProtocolo.Items[j - 1].Selected = true;
                    index = j - 1;
                }
                if(cantidad == 0)
                {
                    index = -1;
                    form.btnEliminar.Enabled = false;
```

Figura 3.16 Ejemplo de segmento de código sin usar los componentes (71 líneas de código).

```
                form.textCodigo.Enabled = false;
                form.textNombre.Enabled = false;
            }
            form.listProtocolo.Focus();
            if(index != -1)
            {
                form.textCodigo.Enabled = true;
                form.textNombre.Enabled = true;
                form.textCodigo.Text = (form.listProtocolo.Items[index].Tag
                    as Protocolo).Codigo;
                form.textNombre.Text = (form.listProtocolo.Items[index].Tag
                    as Protocolo).Nombre;
            }
            else
            {
                form.textCodigo.Clear();
                form.textNombre.Clear();
            }
        }
        catch(Exception ex)
        {
            MostrarExcepcion(ex);
        }
    }
    else
    {
        form.listProtocolo.Focus();
    }
}
banderaNuevo = false;
banderaCambio = false;
form.btnGuardar.Enabled = false;
}
```

Figura 3.17 Ejemplo de segmento de código sin usar los componentes (Continuación) (71 líneas de código).

```
        boton eliminar
private void btnEliminar_Click(object sender, EventArgs e)
{
    frmProtocolo form = (frmProtocolo)Formulario;
    int cantidad;
    frmConfirmar confirmar = new frmConfirmar();
    confirmar.Mensaje = "Está seguro que desea eliminar este elemento";
    confirmar.Encabezado = "Alerta";
    DialogResult verificar = confirmar.ShowDialog();
    if(verificar == DialogResult.OK)
    {
        GtrProtocolo gtrProtocolo = new GtrProtocolo();
        int j = form.listProtocolo.SelectedIndices[0];
        try
        {
            gtrProtocolo.Eliminar(form.listProtocolo.Objeto as Protocolo);
            form.listProtocolo.Del();
            cantidad = form.listProtocolo.Items.Count;
            for(int i = j; i < cantidad; i++)
            {
                form.listProtocolo.Items[i].Text = (i + 1).ToString();
            }
            if(cantidad == 0)
            {
                index = -1;
                form.textCodigo.Enabled = false;
                form.textNombre.Enabled = false;
            }
            form.listProtocolo.Focus();
        }
        catch(Exception ex)
        {
            MostrarExcepcion(ex);
        }
    }
    else
    {
        form.listProtocolo.Focus();
    }
    banderaNuevo = false;
    banderaCambio = false;
    form.btnGuardar.Enabled = false;
}
```

Figura 3.18 Ejemplo de segmento de código usando los componentes (42 líneas de código).

3.12 Conclusiones

En el presenta capítulo concluimos que el uso de NUnit para realizar pruebas de unidad es una herramienta que brinda un nivel de seguridad alto del correcto funcionamiento del sistema a la hora de ser desplegado, lo cual para cualquier sistema en desarrollo es vital. Con el uso de NUnit se gana mucho tiempo ya que las pruebas se realizan de una forma más simple y se pueden conocer los errores que ocurrieron, donde se detectaron; al realizar modificaciones con las pruebas de regresión desarrolladas sobre el mismo nos detallaran si ese cambio afectó el resto del sistema.

CONCLUSIONES

El desarrollo del sistema de Digitalización se logró gracias a una buena selección de las tecnologías informáticas a utilizar y del estudio realizado por parte del equipo de trabajo sobre las necesidades del cliente, lo que permitió el cumplimiento de los objetivos trazados.

- Se ha elevado el control de la información manejada en los Registros y Notarías del MIJ de Venezuela.
- Se creó una nueva herramienta que permite el escaneo de la información que se encuentra en hojas de papel.
- Se permite controlar la calidad de la información escaneada, dando la posibilidad de rechazarla si no cumple con la calidad requerida.
- Permite realizar un seguimiento de las operaciones realizadas sobre los tomos.

Con la implantación de este sistema se lograra digitalizar toda la información almacenada en más de 12 millones de tomos, lo cual aumentara la velocidad de búsqueda y la capacidad de respuesta de las oficinas de los Registros y Notarías del MIJ de Venezuela, dando lugar a la prestación de un mejor servicio.

RECOMENDACIONES

Queremos recomendar al resto de los programadores que aprendan de nuestras buenas prácticas de programación, que aprendan de nuestros errores. Recomendar además el uso de los componentes desarrollados para facilitar la programación, eliminando así las validaciones en tiempo de ejecución, aumentando la claridad del código, cuestiones que se demostraron en las recomendaciones del capítulo 3, su uso es muy ventajoso para cualquier aplicación desktop desarrollada sobre VS.Net.

BIBLIOGRAFÍA

1. **Batista, María Belén Melián.** http://webpages.ull.es/users/estinv/. *Departamento de estadística, investigación operativa y computación.* [En línea] 25 de 2 de 2005. [Citado el: 20 de 4 de 2007.] http://webpages.ull.es/users/mbmelian/TEMA6.pdf.

2. **Booch, Grady.** *Análisis y diseño orientado a objetos 2a edición.* 1996.

3. **Kicillof, Nicolás.** Programación Orientada a Aspectos (AOP). *www.microsoft.com.* [En línea] [Citado el: 20 de 4 de 2007.] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art152.asp.

4. **Quintero, Antonia Mª Reina.** *Visión General de la Programación Orientada a Aspectos.* Sevilla : s.n., 2000.

5. Terminología y Factibilidad de uso para Java. *www.osmosislatina.com.* [En línea] 7 de 9 de 2005. [Citado el: 20 de 4 de 2007.] http://www.osmosislatina.com/java/basico.htm.

6. .NET - Wikipedia, la enciclopedia libre. *Wikipedia.* [En línea] 17 de 4 de 2007. [Citado el: 20 de 4 de 2007.] http://es.wikipedia.org/wiki/.NET.

7. Información general del producto . *www.microsoft.com.* [En línea] 13 de 1 de 2003. [Citado el: 20 de 4 de 2007.] http://www.microsoft.com/spanish/msdn/netframework/productinfo/overview.asp.

8. LEAD Technologies Inc. Corporate Overview. *LEAD Technologies Inc.* [En línea] [Citado el: 20 de 4 de 2007.] http://www.leadtools.com/Home2/press/corporateovr.htm.

9. **Cabrera, Martín.** Introducción a NHibernate. *mcabrera.datacenter1.com.* [En línea] 6 de 2005. [Citado el: 20 de 4 de 2007.] http://mcabrera.datacenter1.com/articles/dotNET/nhibernate/.

10. **H, Joel Francia.** Desarrollo de una Aplicación en tres Capas con VS .NET. *www.microsoft.com.* [En línea] [Citado el: 1 de 5 de 2007.] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art140.asp.

11. **Milanés, Ing. José Manuel.** Midiendo la Complejidad del Software. *www.gruposeti.com.* [En línea] [Citado el: 24 de 5 de 2007.] http://www.gruposeti.com/complejidad.htm.

12. INGENIERÍA DEL SOFTWARE I curso 2006-2007. *http://alarcos.inf-cr.uclm.es.* [En línea] 19 de 4 de 2003. [Citado el: 24 de 5 de 2007.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

GLOSARIO

Autenticarse

Es el proceso por el cual debe pasar todo usuario para entrar en el sistema.

Buscar Tomo por Criterio

Es el proceso que se realiza cuando es necesaria una búsqueda del Tomo para mostrar un resultado esperado.

Circuito

Un circuito es una organización geográfica que contiene un conjunto de Oficinas pertenecientes a una localidad.

Estados

Los estados son parte de cada tipo de movimiento que se realiza en el centro, cada movimiento realiza un cambio de estado al tomo. En el paso del tomo por el centro este puede tomar diversos estados en dependencia de la secuencia por la que transite.

Entre los estados posibles del tomo se encuentran: recepcionado, por desencuadernar, desencuadernado, por escanear, escaneándose, digitalizado; reescanear, revisado, por encuadernar, por devolver y devuelto.

Código de barra

Los códigos de barra son para identificar automáticamente objetos del proceso y facilitar el ingreso de información, eliminando la posibilidad de errores en la captura.

En este caso se utilizan fundamentalmente para dar seguimiento a las operaciones realizadas por los usuarios en los diferentes puestos de trabajo, y dar seguimiento a los tomos, así como controlar a los transportistas involucrados en el proceso.

Oficina

Una Oficina es un Local del Registro Inmobiliario al que pertenece el Tomo.

Protocolo de Tomo

Protocolo al que pertenece el Tomo (se supone que, en los primeros 25 millones de páginas que se digitalicen, todos pertenezcan al Protocolo 1).

Puestos de Trabajo

Son los puestos en los que se realizarán movimientos de tomos dentro del centro.

Lote

El Lote es un paquete de varios Tomos, surge en el proceso de entrada al centro del acta de entrega-recepción.

Metadatos

Son los datos asociados al Tomo a digitalizar que lo identifican, entre ellos podemos mencionar: Protocolo, Oficina, Año, Trimestre, Número del tomo.

Monitorear Digitalización

El monitoreo de la digitalización es tener una estadística de algún dato en específico como: tiempo del Tomo en algún estado, cantidad de Tomos en un estado.

Tipos de dispositivos

Los Tipos de dispositivos son aquellos en se realizará las salvas de la información para su posterior envío a Cuba.

Tipos de Movimientos

Los Tipos de Movimientos son los posibles movimientos que puede adoptar un tomo por la secuencia que se defina en el sistema.

Tipos de Oficina

Son los distintos tipos de oficinas que están definidas para agrupar un conjunto de oficinas a las que puede pertenecer un tomo físico, por ejemplo: las Oficinas inmobiliario, mercantil.

Tomo

Libro que se va a digitalizar, posee acta de apertura y de cierre con datos que precisan tiempo de creación y cierre entre otras informaciones.

Transportista

Son los responsables de realizar operaciones de traslado del tomo tanto físico como digital, hacia el centro y fuera del centro.

Usuario

Un Usuario es aquella persona que interactúa con el Sistema.

Configuración del escáner

Es una configuración específica que va a tener el escáner para la digitalización de los Tomos, esto se lleva a cabo por el administrador del sistema.

Configuración general

Aquí se recogen todas las configuraciones que se van a hacer en el sistema, como: tipos de dispositivos donde se va a guardar la información para enviar a Cuba, el % de revisión de páginas por parte del puesto de calidad, la configuración del escáner. Todo esto se lleva a cabo por el administrador del sistema.