Universidad de las Ciencias Informáticas

Facultad 3

Centro de Gobierno Electrónico

Sistema para la Gestión de la información y Control de Libros v 2.0 para la librería de la Universidad de las Ciencias Informáticas

Trabajo final presentado en opción al título de
Ingeniero en Ciencias Informáticas

Autor: Dayana Donatien Morejón

Tutor: Ing. Reinier Silverio Figueroa

Consultante: MsC. Iliannis Pupo Leyva

La Habana, junio de 2015

"Año del 57 Aniversario de la Revolución"

DECLARACIÓN JURADA DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Dayana Donatien Morejón

_____

Firma del autor(a).

Ing. Reinier Silverio Figueroa

_____

Firma del tutor (a)

Msc. Iliannis Pupo Leyva

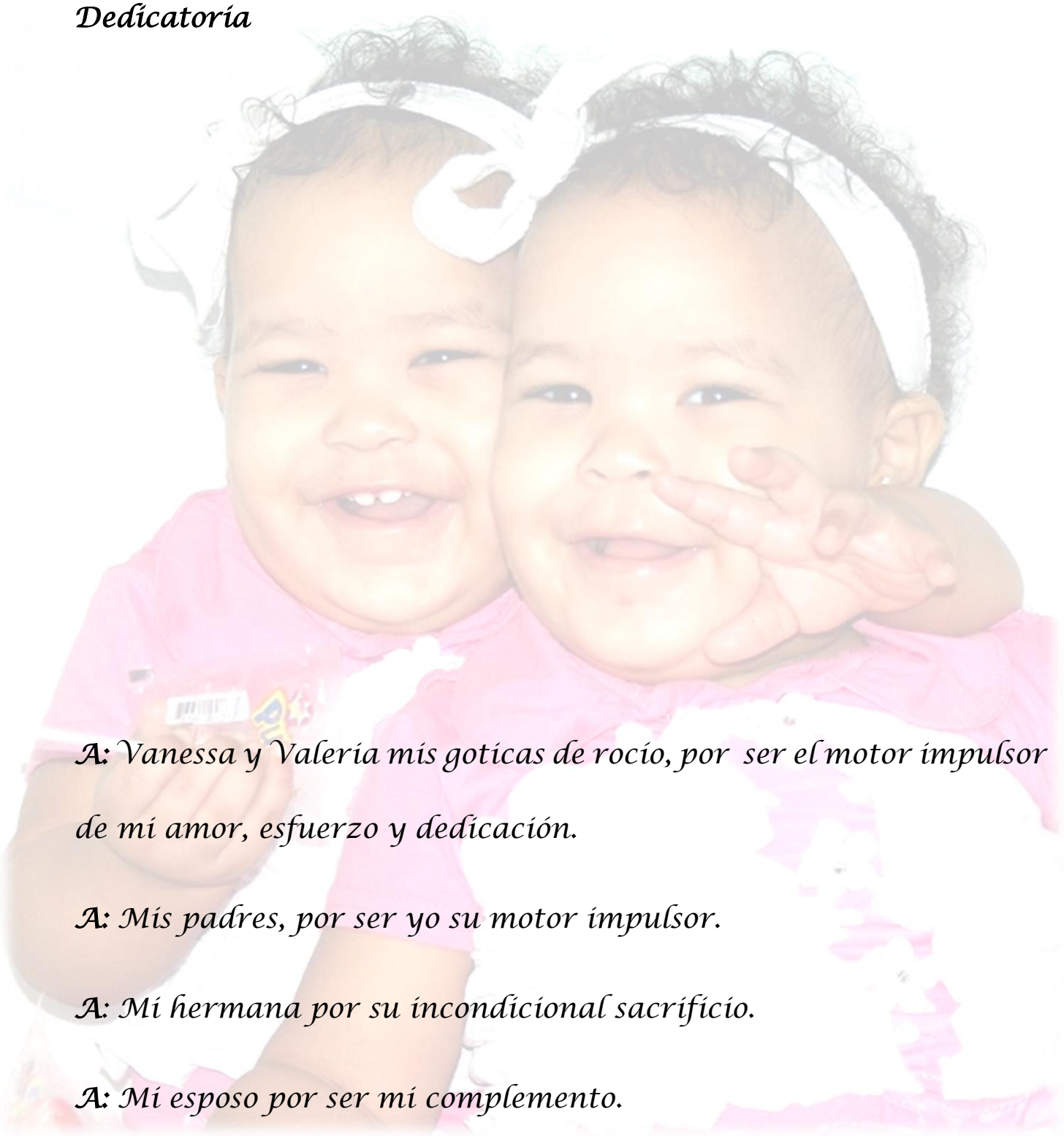
_____

Firma del (la) consultante

Agradecimientos

Agradezco a nuestro Creador, por poner en mi vida la existencia de tantas personas dispuestas a acompañarme en los buenos y no tan buenos momentos. Por darme tantas oportunidades de crecer y hacerme fuerte ante sus pruebas. Por proveerme el aliento necesario cuando sentía perderlo. Por darme la oportunidad de llegar hasta aquí y demostrarme que apenas inicia mi largo camino.

Dedicatoria



A: *Vanessa y Valeria mis goticas de rocío, por ser el motor impulsor de mi amor, esfuerzo y dedicación.*

A: *Mis padres, por ser yo su motor impulsor.*

A: *Mi hermana por su incondicional sacrificio.*

A: *Mi esposo por ser mi complemento.*

A: *Mi suegra por su dulzura, comprensión y apoyo.*

RESUMEN

Para la librería de la Universidad de Ciencias Informáticas, se implementó un sistema informático que permitiera gestionar la información referente a la gestión y control de la literatura que se oferta a la comunidad universitaria. Mediante el estudio de las principales características tecnológicas y funcionalidades implementadas, responsables de dar cumplimiento a los requisitos del cliente, se detectaron aspectos que podrían ser mejorados, en tributo a incrementar el nivel de usabilidad para el usuario y mantenibilidad para el desarrollador. Se valoró además, la posibilidad de gestionar las trazas que describen los principales movimientos que se realizan con los libros.

El presente trabajo, refleja el proceso de desarrollo de una nueva versión del Sistema para la Gestión de la información y Control de Libros, que permita erradicar las deficiencias detectadas en su estudio. Para cumplir este objetivo, se investiga sobre las herramientas y tecnologías, que podrían utilizarse en la implementación de la solución al problema planteado; y se definen las funcionalidades que serán agregadas en esta versión. Como guía del proceso de desarrollo del software, se utiliza una de las metodologías existentes, documentando en cada una de sus fases, los artefactos que propone y otros que no; donde la utilización de estos últimos fue necesaria como buena práctica de la Ingeniería de Software, para describir algunos elementos fundamentales del proceso de desarrollo de la solución .En la última fase de la metodología utilizada, se ejecutan pruebas y métricas de calidad de software, para validar el cumplimiento del objetivo general de la investigación.

Palabras clave: librería, sistema para el control de libros, herramienta, tecnología.

ÍNDICE

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

INTRODUCCIÓN

En la actualidad las librerías como instituciones comercializadoras de literatura manejan un gran volumen de información referente en su mayoría a los ejemplares, objetos de la venta al público. Para llevar a cabo el control sobre la información de cada libro y atendiendo al auge y amplia utilización de las Tecnologías de la Información y las Comunicaciones (TIC), surgen sistemas informáticos que agilizan los procesos de inventarios y permiten conocer en todo momento cual es la situación en cada área teniendo en cuenta categorías, así como otros aspectos discriminantes para clasificar la literatura comercializada. En la Universidad de las Ciencias Informáticas (UCI) se cuenta con una librería que brinda sus servicios a la comunidad universitaria. Para ésta institución se han realizado varias investigaciones con el objetivo de informatizar sus procesos inventariales, en la búsqueda de centralizar la información, evitar el deterioro de los archivos físicos y facilitar los movimientos internos de los libros (Suarez, 2012). La investigación más reciente tuvo como resultado un Sistema para la Gestión de Información y Control de libros, SGCL por sus siglas en español, elaborado sobre la Plataforma Estándar de Java (JSE por sus siglas en inglés) en su versión 1.6.26, utilizando como gestor de base de datos PostgreSql en su versión 9.1. Se empleó además bibliotecas de interfaces gráficas como SWING y SWINGX, ambas vinculadas a la plataforma de Java antes mencionada. Se realizó un análisis de la solución mencionada que permitió identificar aspectos nuevos o mejorables tanto tecnológicos como funcionales relacionados a continuación:

- No se provee la posibilidad de auditar los movimientos realizados en un periodo de tiempo determinado.
- La gestión, referida a la actualización y visualización, de las fuentes de datos persistentes, se realiza de una forma compleja desde el punto de vista de la programación, aspecto que puede dar al traste con la presentación de los datos en dependencia de sus características.
- Existencia de niveles de acoplamiento entre las clases de interfaces de usuario no deseados, lo que dificulta el mantenimiento y la reutilización de las mismas.
- Arquitectura del sistema distinguida por un estilo Modelo-Vista-Presentador, donde la mayor parte de la lógica de negocio se presenta en la vista, aspecto que trae como consecuencia dificultades en la reutilización y la gestión de cambios en el negocio.
- Limitaciones en cuanto a la usabilidad del sistema, debido al uso excesivo de cuadros de diálogo en la presentación del flujo de eventos, sin aprovechamiento del área de trabajo.
- Obsolescencia del JDK (Java Development Kit) utilizado, que propicia la existencia de bugs ya solucionados en versiones actuales y no soporta el uso de la biblioteca JavaFX para la construcción de interfaces gráficas de usuario.
- Uso del Sistema de Gestión de Bases de Datos PostgreSql que, aunque ofrece potencialidades ampliamente demostradas para el almacenamiento de datos y las transacciones sobre estos, debido a las características de uso del sistema presenta desventajas frente a gestores embebidos, debido

a las necesidades de mayores prestaciones físicas y seguridad que requiere y con las que hoy no cuenta la institución.

Una vez analizado lo planteado con anterioridad se identifica el siguiente **problema a resolver**: Las limitaciones referidas a la implementación de los requisitos funcionales y no funcionales del Sistema para la Gestión de Información y Control de libros en su versión 1.0 afectan su mantenibilidad y usabilidad.

A raíz de dicho problema se define como **objeto de estudio** los procesos de gestión y control de libros en la librería de la Universidad de las Ciencias Informáticas.

Para solucionar el problema planteado se define como **objetivo general**: Desarrollar la versión 2.0 del Sistema para la Gestión de Información y Control de libros que permita mejorar las características de mantenibilidad y usabilidad a partir de la sustitución de tecnologías de desarrollo, implementación de nuevas funcionalidades y mejora de otras ya existentes.

Para la descomposición del objetivo general se definen los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación.
- Rediseñar el sistema de acorde a las nuevas tecnologías a emplear.
- Implementar el diseño elaborado para el sistema.
- Validar la solución propuesta.

Para englobar el contenido de este trabajo se define el siguiente **campo de acción**: Sistema para la Gestión de Información y Control de libros en la librería de la Universidad de las Ciencias Informáticas.

Idea a defender: Con el desarrollo de la versión 2.0 del Sistema para la Gestión de Información y Control de libros se logrará eliminar las limitaciones actuales referidas a los requisitos funcionales y no funcionales que afectan su mantenibilidad y usabilidad.

En aras de dar cumplimiento a los objetivos específicos de la investigación se definen las siguientes tareas investigativas:

- Estudio de los procesos de negocio de la librería de la Universidad de las Ciencias Informáticas.
- Estudio del Sistema para la Gestión de Información y Control de libros.
- Estudio y selección de las herramientas y tecnologías de desarrollo para el desarrollo de la solución.
- Estudio y selección de la metodología de desarrollo a utilizar.
- Describir las Historias de usuario referentes a las funcionalidades del sistema.
- Especificación de los requisitos no funcionales del sistema.
- Diseñar el modelo de datos con la incorporación de nuevos requisitos.
- Realización de las tareas de implementación.
- Verificación del sistema a partir de la aplicación de métricas de requisitos, métricas de diseño, pruebas unitarias y funcionales.
- Validación de las variables de investigación.

Los **métodos científicos** a emplear para guiar el estudio del entorno de la investigación y darle cumplimiento a estas tareas son:

Métodos teóricos

- Análisis de la literatura (Mikael Berndtsson, y otros, 2008):
  Utilizado para; a partir del análisis tecnológico y funcional del sistema a mejorar; seleccionar la literatura indicada, consultadas sobre las aplicaciones para el control de libros, así como de las herramientas y las tecnologías que serán utilizadas, obtener los elementos significativos y arribar a conclusiones que permitan dar solución al problema.
- Modelación (Rolando Alfredo Hernández León, 2011):
  Empleado para la creación de los modelos de negocio, diseño de bases de datos y de las clases que serán utilizados para el desarrollo de la solución.

Métodos empíricos

- Medición:
  Se utiliza con el objetivo de obtener una medida para los parámetros de calidad y variables del problema de investigación, que permita verificar que el proceso desarrollo de la solución marcha correctamente y que al culminarlo se obtenga el producto deseado.
- Implementación (Mikael Berndtsson, y otros, 2008):
  Empleado para realizar la implementación de todo el diseño modelado para la solución, permitiendo materializar la satisfacción de la necesidad de la investigación.

El documento se estructura en 3 capítulos que recogen todo lo relacionado con el trabajo investigativo realizado, así como la solución al problema planteado.

El **Capítulo 1 Fundamentación Teórica**, está dedicado al estudio del Sistema para la Gestión de la información y Control de Libros, implementado para la librería de la UCI enfocado a sus características funcionales y tecnológicas. Se explica cuales elementos mejorar o cambiar como solución al problema planteado, así como las tecnologías, herramientas y metodologías a utilizar para su desarrollo.

En el **Capítulo2 Análisis y Diseño,** desde un primer momento se describen los requisitos funcionales y no funcionales, estos últimos a través de la definición de las Historias de Usuarios. Se evidencia como mediante el diagrama de clases del diseño se muestran las clases necesarias para la informatización del proceso de gestión de libros, cuya implementación se realiza tomando como guías de diseño patrones GRASP y GoF. Seguidamente se refleja la utilización de las Tarjetas Clase-Responsabilidad-Colaboración (CRC), para la representación de las relaciones entre las mismas. Se diseña también el modelo de datos que permite describir la estructura de los mismos y la forma en que se relacionan. Se demuestra como con el uso de un patrón arquitectónico se provee un esquema organizativo estructural, fundamental para el sistema de software que se propone como solución. Por último se reflejan los elementos necesarios a tener en cuenta en el momento de la codificación.

El **Capítulo 3. Implementación y Prueba** se dedica a la verificación y validación del sistema implementado como solución, mediante pruebas unitarias y funcionales realizadas al producto, mostrando sus resultados en cada caso permitiendo constatar la solución propuesta.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 La usabilidad y mantenibilidad como atributos de calidad de los sistemas informáticos

Partiendo de la problemática planteada anteriormente, es importante conocer y comprender a priori la definición de los conceptos usabilidad y mantenibilidad como atributos de calidad de los sistemas informáticos. Estos están concebidos entre los seis atributos claves que identifica el estándar ISO 9621 (Roger S. Pressman, 1997).

Usabilidad

Constituye el esfuerzo necesario para aprender a operar con el sistema, preparar los datos de entrada e interpretar las salidas de un programa. Si un programa no es fácil de usar, frecuentemente estará abocado al fracaso, incluso aunque las funciones que realice sean valiosas. La facilidad de uso es un intento de cuantificar lo amigable que puede ser el sistema con el usuario (Roger S. Pressman, 1997). La usabilidad es una cualidad demasiado abstracta como para ser medida directamente. Para poder estudiarla se descompone habitualmente en los siguientes cinco atributos básicos (Grau, 2000):

- Facilidad de aprendizaje: Cuán fácil es aprender la funcionalidad básica del sistema, como para ser capaz de realizar correctamente la tarea que desea realizar el usuario. Se mide normalmente por el tiempo empleado con el sistema hasta ser capaz de realizar ciertas tareas en menos de un tiempo dado.
- Eficiencia: El número de transacciones por unidad de tiempo que el usuario puede realizar usando el sistema. Lo que se busca es la máxima velocidad de realización de tareas del usuario. Cuanto mayor es la usabilidad de un sistema, más rápido es el usuario al utilizarlo, y el trabajo se realiza con mayor rapidez.
- Recuerdo en el tiempo: Para usuarios intermitentes (que no utilizan el sistema regularmente) es vital que sean capaces de usar el sistema sin tener que aprender cómo funciona partiendo de cero cada vez. Este atributo refleja el recuerdo acerca de cómo funciona el sistema que mantiene el usuario, cuando vuelve a utilizarlo tras un periodo de no utilización.
- Tasa de errores: Este atributo contribuye de forma negativa a la usabilidad de un sistema. Se refiere al número de errores cometidos por el usuario mientras realiza una determinada tarea. Un buen nivel de usabilidad implica una tasa de errores baja. Los errores reducen la eficiencia y satisfacción del usuario, y pueden verse como un fracaso en la transmisión al usuario del modo de hacer las cosas con el sistema.
- Satisfacción: Éste es el atributo más subjetivo. Muestra la impresión subjetiva que el usuario obtiene del sistema.

La usabilidad del sistema no es una simple adición del valor de estos atributos, sino que se define para cada sistema como un nivel a alcanzar para algunos de ellos.

Relación entre la Usabilidad y la Interfaz Gráfica de Usuario

En el desarrollo de software se identifica a menudo la usabilidad con las características de los elementos de una interfaz gráfica de usuario basada en ventanas, como puede ser su color, su disposición o el diseño gráfico de los iconos y animaciones. Sin embargo, la usabilidad no sólo tiene que ver con la interfaz gráfica de usuario.

La usabilidad de un sistema está ligada principalmente a la interacción del mismo, al modo en que se realizan las operaciones con el sistema. Esta interacción no está definida en la interfaz gráfica, sino que está imbricada en el código que implementa la funcionalidad del sistema. La interfaz gráfica de usuario es la parte visible de tal interacción. Es cierto que esta es una parte importante del sistema, y un buen diseño de la misma puede propiciar altos niveles de usabilidad, pero un sistema con un diseño de la interacción pobre no puede mejorar su nivel de usabilidad tan solo cambiando la interfaz gráfica. La forma de implementación de la visualización y presentación de los datos al usuario, es entre otros factores influyentes, uno de los elementos que más pudiera afectar la obtención de niveles aceptables de usabilidad para el producto final.

Mantenibilidad

Facilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia, o mejorar si el cliente desea un cambio de requisitos (Roger S. Pressman, 1997).

Según la ISO – 25010 (ISO, 2015), los atributos relacionados a continuación permiten evaluar la mantenibilidad de un sistema informático de una forma más concreta:

- Modularidad: Representa el grado en que un sistema o programa se estructura por componentes que al ser modificados no generan impacto en otros componentes, determinado por el nivel de acoplamiento que pueda existir entre ellos.
- Reusabilidad: Representa el grado en que un activo puede ser utilizado en más de un sistema de software o dentro del mismo.
- Analizabilidad: Representa la facilidad para analizar el software en busca de deficiencias e identificar sus componentes.
- Capacidad de cambio: Responde a la capacidad y facilidad para realizar cambios en el software.
- Estabilidad: Capacidad para evitar efectos inesperados tras realizar modificaciones en el software.
- Capacidad de pruebas: Representa la capacidad para validar los cambios del software.
- Cumplimiento de mantenibilidad: Grado en que el producto de software se adhiere a las normas o convenciones relacionadas con la mantenibilidad.

Estos atributos pueden ser revisados aplicando métricas sobre las características internas y externas del sistema.

1.2 Proceso para la gestión de información y control de libros en la Universidad de las Ciencias informáticas

El término librería es utilizado para definir a aquellas instituciones en las que el principal producto de venta son los libros, aunque los mismos pueden estar complementados por revistas, diarios y por otros materiales multimedia como CD o videos. Una librería puede variar no sólo en su tamaño sino también en lo que respecta al tipo de libros que se vende o al tipo de atención. Normalmente, a diferencia de lo que sucede en las bibliotecas, en una librería la persona puede recorrer las diferentes estanterías y seleccionar el libro que es de su interés para luego abonarlo en el sector de caja. (Galilea Rhode, 2013) En todas las librerías es necesario que se registre un control de todos los libros que entran, salen y existen en ellas. La librería de la UCI está compuesta por: el salón donde se le oferta la literatura a los clientes, el almacén donde se recibe y se conserva la literatura que será ofertada y la oficina de economía donde se registran las operaciones diarias realizadas con la literatura. En todas las librerías el proceso de control de la literatura se realiza mediante el manejo de la siguiente información:

- Informes de Recepción: Documento en el que se registran todos los datos de los libros que son recibidos de la distribuidora.
- Facturas por Cheques: Esta Factura se genera cuando hay alguna compra a través de cheque.
- Vales de Venta: Es el comprobante que se le entrega al cliente una vez que ha pagado su compra, donde se refleja el importe de la misma.
- Vales de Salida: Este documento se genera cuando se abastece el salón.
- Vale de Devolución: Se realiza cuando se devuelven libros del salón al almacén por una determinada razón.
- Reporte Diario de Operaciones: Documento es elaborado por la económica al culminar las ventas del día, recogiendo el importe de dichas ventas para llevar a cabo el depósito que se debe realizar diario al banco.
- Modelo de Inventario de Venta: Es realizado por la dependienta al culminar las ventas del día para llevar un control detallado de los libros vendidos y lo que se ha recaudado.
- Tarjeta de Estiba Almacén: Se realiza para llevar un control de los movimientos del libro en el almacén.
- Tarjeta de Estiba Salón: Se realiza para llevar un control de los movimientos del libro en el salón.

Se realizó un proceso de búsqueda e investigación en la librería de la UCI, con el objetivo de describir el proceso de control de los libros como se muestra en la Ilustración 1. Este proceso es inicializado cuando el almacenero efectúa la recepción de literatura. Luego la económica realiza el control de la literatura interna mediante el uso del sistema, donde se realizan los inventarios y se generan los reportes de las operaciones.

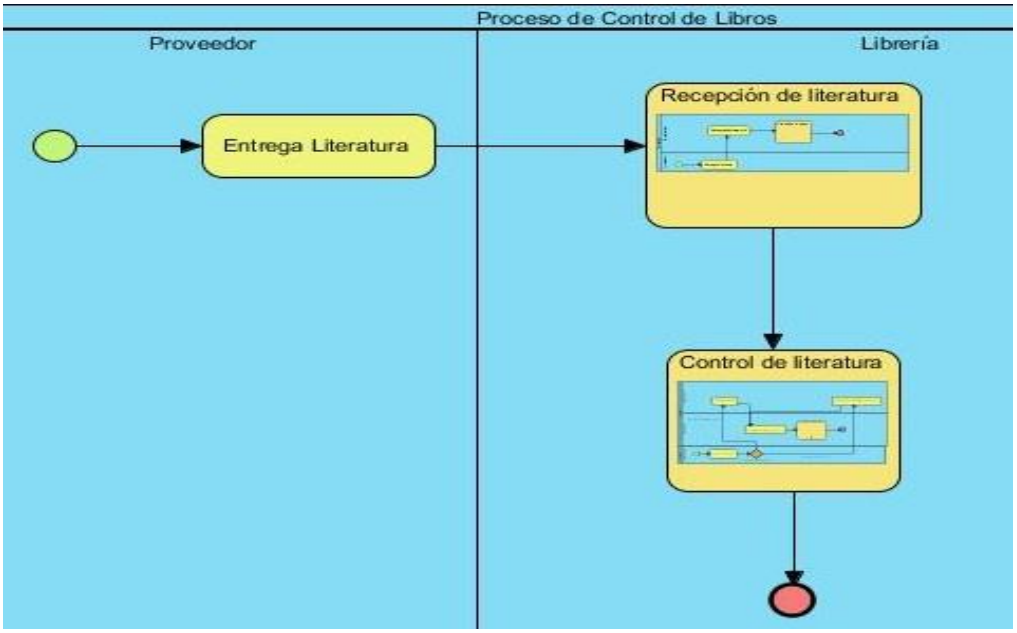


Ilustración 1: Proceso de Control de Libros de la librería de la UCI.

Dentro los principales procesos que se realizan en la librería de la UCI se identifica la Recepción de Literatura. Mediante el registro de los datos de cada libro recibido por el almacenero, se gestiona su información generando en esta actividad el Informe de Recepción. (Ver ilustración 2)

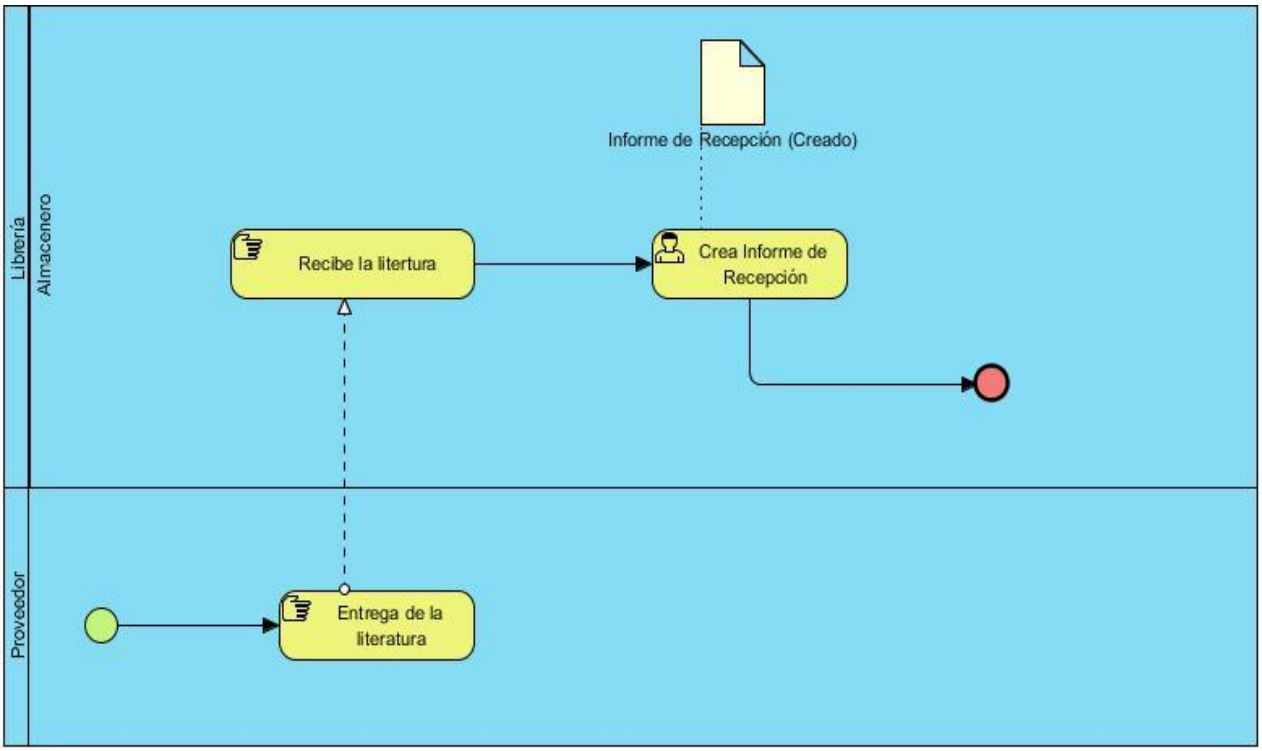


Ilustración 2: Recepción de la literatura.

Otro de los procesos tradicionales realizados en cualquier librería y por tanto en la de la UCI, es la Gestión de Ventas (ver Ilustración 3), donde se le da la atención requerida a cada cliente que se presenta en la librería. En el mismo se genera una factura cuando la venta se realiza a través de un

cheque. Los vales de las ventas se le entregan al cliente una vez que han pagado su compra reflejando el importe de las mismas. Al finalizar el día se genera un resumen o modelo de venta realizado por la (el) dependienta (e) para llevar un control detallado de los libros vendidos y el monto que se ha recaudado.(Ver Ilustraciòn 3)

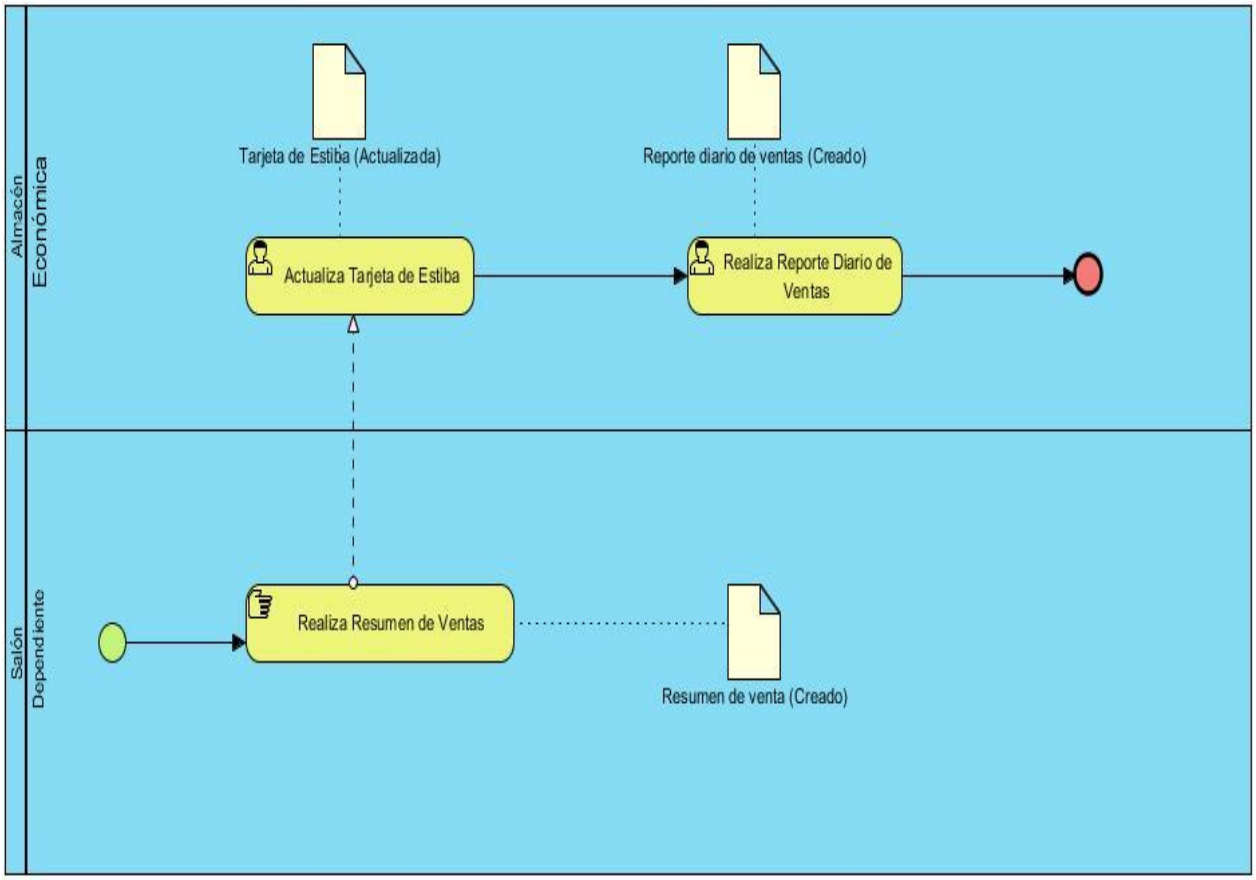


Ilustración 3: Gestión de ventas.

También se realiza el Control de literatura existente en la librería, donde se tiene en cuenta el tipo de solicitud que realice el (la) dependiente (a) al almacenero (a). Cuando el primero realiza una solicitud, si es de abastecimiento el almacenero abastece el salón y crea un Vale de Salida; si la solicitud es de devolución entonces se recepciona la literatura y el vale creado es de devolución, actualizándose en cada caso las tarjetas de estiba del almacén y el salón. (Ver Ilustración 4)

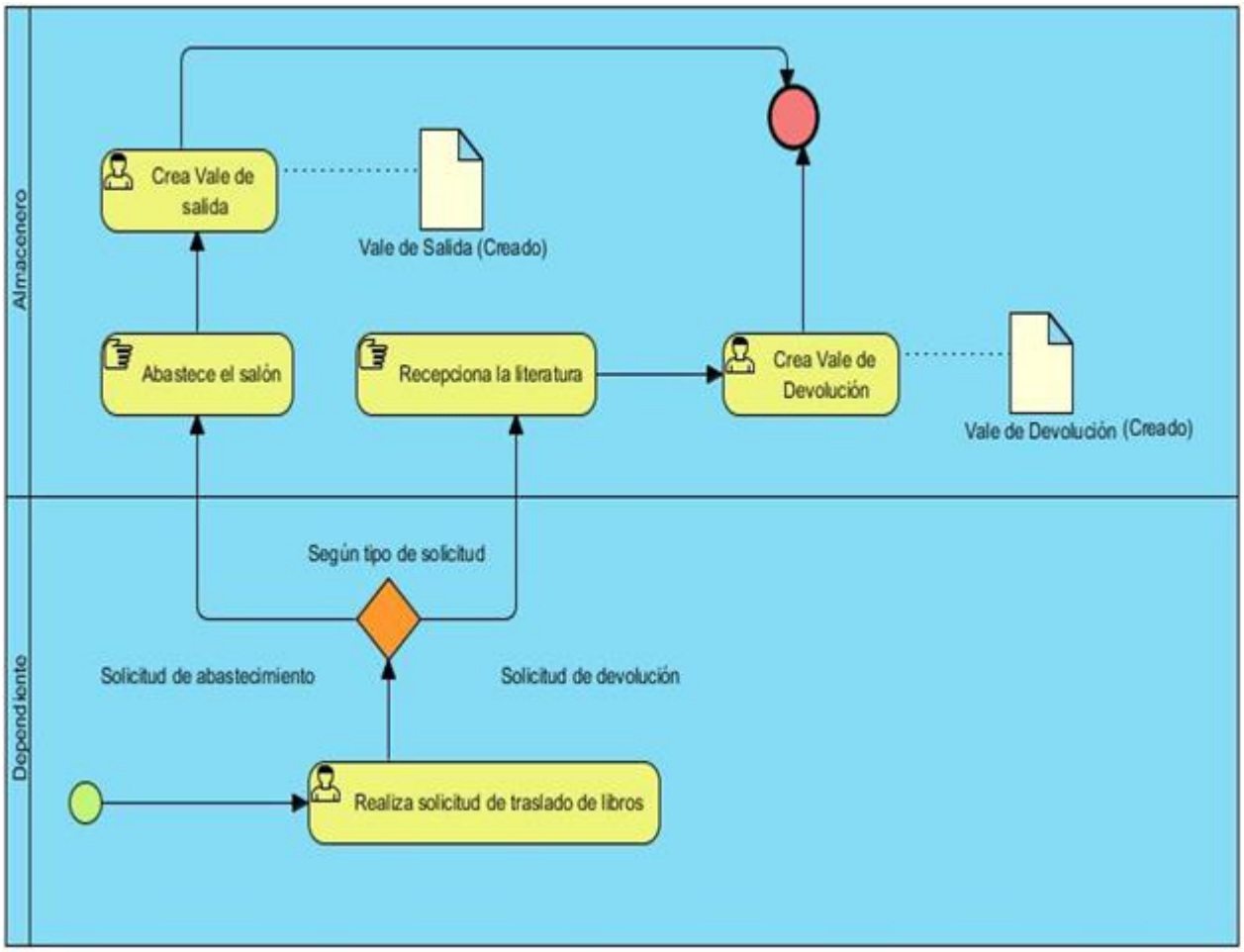


Ilustración 4: Control de la literatura.

Un proceso realizado en la librería que no puede dejar de mencionarse es la Gestión de Informes, encargado de generar la documentación que se maneja en la librería. Al finalizar el día la dependienta en interacción con el sistema, es la encargada de realizar el resumen diario de ventas para a partir del mismo la económica pueda crear el reporte diario de ventas, actualizándose automáticamente las tarjetas de estiba y de salón de cada literatura manejada (ver Ilustración 3). Por su parte el almacenero cuando recepciona la literatura entregada por el proveedor, crea un informe de recepción y si la literatura existe, el sistema actualiza la Tarjeta de Estiba de almacén de la misma; sino la crea (ver Ilustración 2).

Finalmente se realiza la Gestión de los recursos humanos que se ocupa de seleccionar, contratar, capacitar y emplear a los colaboradores de la empresa, así como velar porque cuenten con los medios necesarios para desarrollar sus funciones.

Con el sistema en estudio solo se informatizaron, dentro del proceso de control de los libros los subprocesos de: Recepción de literatura, Control de literatura y Gestión de informes. La presente investigación, pretende incorporar funcionalidades responsables de la gestión de los movimientos de la información que se genera en cada operación dentro del sistema, relacionada con documentos y usuarios.

1.3 Análisis del sistema existente

Se realizó un estudio del Sistema para la Gestión de la información y Control de Libros existente en la Universidad de las Ciencias Informáticas, persiguiendo dos objetivos fundamentales:

- Determinar un mecanismo para que las trazas generadas por los procesos del sistema, puedan ser auditables en un periodo de tiempo determinado.
- Describir las limitaciones referidas a la implementación de los requisitos funcionales y no funcionales, que inciden significativamente en su mantenibilidad y usabilidad.

1.3.1 Descripción funcional del sistema

El sistema existente es una herramienta para la gestión de información y control de los libros en la librería de la Universidad de las Ciencias Informáticas. Las principales funcionalidades están orientadas a:

- La gestión de literatura: Posibilita el acceso y control a la literatura existente en la librería.
- La gestión de reportes: Encargada de la generación y visualización de la documentación manejada para realizar el control de los libros.

La información obtenida a partir de la ejecución de estas funcionalidades, es presentada en su mayoría mediante el uso de cuadros de diálogos independientes de la interfaz de usuario correspondiente al marco principal del sistema.

Por otra parte, se realiza el tratamiento de los errores con el fin de garantizar la confiabilidad e integridad de la información que se manipula. Por ejemplo antes de realizar cualquier acción se validan los datos a nivel de interfaz, antes de eliminar se le pide confirmación al usuario. Se advierte también cuando no ha ejecutado correctamente las acciones, manteniéndolo todo el tiempo informado de los errores que se pudieran cometer en el manejo del sistema, por ejemplo el dejar campos vacíos. Se garantiza el nivel de acceso de los usuarios a la información mediante la autenticación. Además se trabaja con variables de sesión lo que posibilita que determinada información presente en la aplicación solo se muestre a los usuarios que tienen autorización previa, de esta forma se delimitan y se preservan los datos permitiendo una mayor seguridad de la información.

Dentro de la descripción antes realizada, hay aspectos que podrían modificarse o adicionarse con el objetivo de mejorar el sistema existente desde el ámbito funcional. Tal es el caso de la utilización rutinaria de cuadros de diálogo para presentar la información al usuario, en lugar de aprovechar el área de trabajo disponible, comprometiendo además la garantía de usabilidad. Podrían adicionarse además, funcionalidades que tengan la responsabilidad de aumentar la seguridad con que dispone el sistema, debido a que carece de auditabilidad, pues no se posee un mecanismo para llevar la trazabilidad de los movimientos realizados en un periodo de tiempo determinado.

1.3.2 Descripción tecnológica del sistema

Se analizaron los aspectos tecnológicos involucrados en la implementación de los requisitos del sistema, en búsqueda de factores que influyen directamente en los atributos de calidad usabilidad y mantenibilidad.

Uso de SWING para el diseño e implementación de las interfaces gráficas de usuario.

Las interfaces gráficas de usuario están diseñadas con el uso de SWING, una biblioteca de componentes gráficos que ha aparecido en la versión 1.2 de Java. SWING fue creado para desarrollar interfaces gráficas en Java con independencia de la plataforma. Está integrado por un amplio conjunto de componentes de interfaces de usuario que funcionan en el mayor número posible de plataformas. Cada uno de los componentes de esta biblioteca puede presentar diversos aspectos y comportamientos en función de una biblioteca de clases. (Frías, 2012)

Es importante destacar que SWING sigue un simple modelo de programación por hilos y que presenta tres características que lo representan:

- Independencia de plataforma lo que significa que su funcionamiento no está determinado por ningún sistema operativo.
- Extensibilidad: es una arquitectura particionada, es decir, los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender de clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Personalizable: dado el modelo de representación programático del marco de trabajo de SWING, el control permite representar diferentes estilos de apariencia " Look and Feel".

Además, los usuarios pueden proveer su propia implementación de apariencia, que permitirá cambios uniformes en la apariencia existente en las aplicaciones SWING sin efectuar ningún cambio al código de aplicación.

La utilización incorrecta de SWING en algunos aspectos en el desarrollo de las interfaces gráficas de usuario del sistema en estudio, trae consigo determinadas desventajas que pueden ser solucionadas:

- Se producen altos niveles de complejidad en la implementación encargada de representar los datos almacenados utilizando el componente visual JTable, el cual carece de un mecanismo que permita insertar en las tablas los objetos como un todo y dificulta habilitar editores especializados en las celdas. Dadas las características de los procesos de negocio de la librería, las tablas fueron utilizadas con frecuencia para la visualización de la información solicitada por el usuario, implicando la repetición de código y por consiguiente complejidad para el mantenimiento.

  En la Ilustración 5 se visualiza el fragmento de código correspondiente a la creación de la tabla encargada de presentar los libros que serán trasladados del almacén al salón de oferta de la literatura. Se observa que, para visualizar los datos de un objeto determinado, se debe recorrer cada columna para asignar los valores correspondientes a sus atributos mediante el llamado a la

base de datos. Se evidencia la necesidad de insertar cada atributo para la fila correspondiente en lugar del objeto completo.

Estos elementos desde el punto de vista de la programación dificultaron la presentación de los datos variables en formato, longitud, tipos y otras características.

```
public void cargarTablaLibrosParaTraslado(boolean databaseRefresh) {
    if (databaseRefresh) {
        librosSeleccionados = LibroManager.getInstance().obtenerLibrosPorInventario(el);
    }

    DefaultTableModel dtm = (ConditionalTableModel) xtableLibrosTraslado.getModel();
    Object[] row = new Object[dtm.getColumnCount()];
    dtm.setRowCount(0);
    for (LibroObjeto libroObjeto : getListaLibrosSeleccion()) {
        row[0] = updatedRows.containsKey(libroObjeto.getCodigo());
        row[1] = libroObjeto.getCodigo();
        row[2] = libroObjeto.getTitulo();
        row[3] = libroObjeto.getPlan();
        row[4] = updatedRows.containsKey(libroObjeto.getCodigo()) ? updatedRows.get(libroObjeto.getCodigo()) :
                 libroObjeto.getCantidadTraslado();
        row[5] = el.equals(ENUM_LUGAR.ALMACEN)?libroObjeto.getCantidadAlmacen():el.equals(ENUM_LUGAR.SALON)?
                 libroObjeto.getCantidadSalon():libroObjeto.getCantidadAlmacen()+libroObjeto.getCantidadSalon();

        dtm.addRow(row);
    }
}
```

Ilustración 5: Codificación para cargar una tabla con los datos de un objeto.

Arquitectura del sistema

Se realizó un estudio de la estructura arquitectónica del SGCL v1.0 detectándose la utilización del estilo Modelo-Vista-Controlador. Esto indica la existencia de la implementación de gran parte de la lógica del negocio en la capa vista, trayendo como consecuencia afectaciones respecto a la reutilización de las clases y dificultades ante la necesidad de implementar cambios en el negocio.

Acoplamiento entre las clases de Interfaz Gráfica de Usuario

Un análisis realizado al sistema anterior desde la implementación de las interfaces graficas de usuario, permitió percibir que:

- El nivel de acoplamiento entre las clases es desfavorable.
- Las clases poseen una fuerte dependencia entre sí.

Esta situación evidencia la poca reutilización existente entre las clases lo que implica una complejidad de implementación elevada. Otra de la consecuencia se materializa ante la necesidad de implementar

cambios en una de las clases, ya que debido a la fuerte dependencia existente, se afectaran también las que estén vinculadas, implicando que el futuro mantenimiento sea complejo.

Uso de JDK 1.6.26

El JDK es un entorno de desarrollo para crear aplicaciones applets y componentes, utilizando el lenguaje de programación Java, donde se incluyen herramientas útiles para desarrollar y probar programas escritos en dicho lenguaje (Oracle Technology Network, 2014) .La versión más actualizada de JAVA es la 1.8.31, lo que indica que el SCGL v 1.0 posee un JDK obsoleto, imposibilitando la utilización futura de tecnologías más actualizadas , que puedan proveerle mejoras de usabilidad y mantenibilidad.

Uso del Sistema Gestor de Bases de Datos PostgreSql

Un Sistema Gestor de Base de Datos (SGBD, en inglés DBMS: Data Base Management System) permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación. Un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base junto con un grupo de operaciones para manejarlos (Eliza Bertino, 1995).

PostgreSql

Este gestor de base de datos fue seleccionado en la implementación de la base de datos del sistema existente teniendo en cuenta que:

- Se ejecuta en gran variedad de sistemas operativos: Linux, Unix, Windows, etc.
- Soporta todas las características de una base de datos profesional, funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas, etc.).
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Adaptable a las necesidades del cliente.
- Permite la gestión de usuarios y permisos.

Es cierto que PostgreSql es un gestor de bases de datos potente y robusto, pero relativo a las características del entorno del sistema en análisis, no es netamente necesaria su utilización. El uso de PostgreSql implica mayores prestaciones físicas de la estación de trabajo donde se instala para su correcta explotación, que las que posee la infraestructura tecnológica de la entidad. Dado que el volumen de datos a informatizar es pequeño y no existe el acceso concurrente de usuarios al sistema, es aceptable que las bases de datos se implementen con gestor más sencillo desde el punto de vista de las necesidades de recursos.

Independientemente de que con las características tecnológicas analizadas se hayan dado respuestas a los requisitos funcionales y no funcionales del sistema en su versión 1.0, existen aspectos mejorables y que pueden cambiar. Es necesario utilizar herramientas alternativas que posibiliten incrementar la

usabilidad para el desarrollador durante la implementación y en consecuencia, disminuir la complejidad de su mantenimiento.

1.4 Selección de la solución

Caracterizados el sistema en estudio y las principales herramientas utilizadas para la implementación del mismo, se propone una solución que aporte mejoras tecnológicas y funcionales orientadas a:

- Proveer la posibilidad de auditar el sistema en un tiempo determinado.
- Utilizar tecnologías alternativas para la implementación de las interfaces gráficas de usuario que simplifiquen la codificación utilizada para la implementación, tributando a la menor complejidad de mantenimiento posible.
- Disminuir el acoplamiento entre las clases de la interfaz gráfica de usuario orientado a incrementar la reutilización entre las mismas y la futura mantenibilidad.
- Estructurar la implementación del sistema en la búsqueda de separar la lógica del negocio de la vista, con el objetivo de eliminar dificultades existentes en la reutilización y la gestión de cambios en el negocio.
- Proporcionar un medio de almacenamiento de datos más sencillo que el existente que consuma menos recursos teniendo en cuenta las características de hardware del entorno donde se desplegará la solución.

Se hace necesario estudiar las tecnologías y herramientas que faciliten la solución de las necesidades planteadas anteriormente

1.4.1 JavaFX frente a SWING

JavaFX es una tecnología de código abierto construida sobre la Máquina Virtual Java, desarrollada por Chris Oliver, que solía trabajar en una compañía llamada “See Beyond Technology Corp” y que fue adquirida por Sun / Oracle en el año de 2005. Los desarrolladores de JavaFX crean aplicaciones utilizando JavaFX Script, un lenguaje de programación creado sólo para este propósito, que es una mezcla de técnicas de escritura de codificaciones tradicionales y un nuevo enfoque declarativo para la creación de interfaces gráficas de usuario. (Ernst, 2011) En la actualidad Oracle ha integrado esta tecnología como una API a la edición estándar de Java, facilitando aprovechar las características de las demás APIs que conforman esta edición.

Características de JavaFX:

- Acelera los ciclos de desarrollo, porque proporciona un flujo de trabajo intuitivo que separa la lógica el modelo de interacción y el diseño, que permite realizar prototipos rápidamente, añadiendo flexibilidad al proyecto.
- Bajo coste de implementación. La creación de Interfaces Gráficas de Usuario es rápida, sencilla y potente.
- Plataforma completamente gratuita.
- Bajo consumo de CPU.

- Excelente gestión de memoria. (timo-ernest, 2010)

Durante más de 10 años, los desarrolladores de aplicaciones han encontrado en SWING una herramienta muy eficaz para la creación de interfaces gráficas de usuario (GUI), y añadir interactividad a las aplicaciones Java, sin embargo se ha podido ver que con JavaFX la creación de estas interfaces de usuario demanda menor codificación y el tiempo de desarrollo se disminuye en gran medida. (Oracle, 2013)

Utilizar JavaFX para implementar las interfaces gráficas de usuario del sistema a desarrollar, evita la complejidad en la implementación de las tablas, encargadas de la presentación de la información al usuario, reflejado en la posibilidad de insertar los objetos como un todo. Mediante el uso de FXML, JavaFX permite de forma sencilla y elegante diseñar interfaces de usuario (UI) con un estilo de programación declarativa. También aprovecha toda la potencia de Java, porque se puede instanciar y utilizar las muchas de las clases de Java que existen en la actualidad. Añadir características tales como la unión de las propiedades de la interfaz de usuario en un modelo y change listeners, reduce la necesidad de implementar método setter y permite a los objetos mantener una actualización constante en cada evento que los involucre. Es decir, los componentes gráficos están más sincronizados con el patrón de diseño Observer, donde a partir de la existencia de la clase oyente ChangeListener, se mantiene una constante comunicación entre componentes gráficos y colecciones de objetos de clase ante posibles cambios, lo que facilita la constante actualización entre estos.

Estas condiciones ofrecidas por JavaFX, pueden propiciar mejorías en la gestión referida a la actualización y visualización de las fuentes de datos persistentes, debido a que conllevan a una menor complejidad de programación y favorecen en gran medida la mantenibilidad para el sistema.

1.4.2 Java Derby frente a PostgreSql

Es un Sistema Gestor de Base de Datos Relacional escrito en Java, que puede ser embebido en aplicaciones desarrolladas con este lenguaje, y utilizado para procesos de transacciones en línea. Inicialmente distribuido como IBM Cloudscape, Apache Derby es un proyecto de código abierto licenciado bajo la Apache 2.0 License. Actualmente se distribuye como Sun Java DB.

Características de JavaDB:

- Es muy liviano, dado que emplea cerca de 2 MB para poner en función el motor de la base y el driver JDBC embebido.
- Está basado en Java, JDBC y estándares SQL.
- Provee un driver JDBC que permite embeber a Derby en cualquier solución Java.
- Soporta el modo cliente/servidor.
- Es fácil de instalar, desplegar y usar.
- Manipulación de objetos complejos en forma rápida y ágil.

El utilizar Java Derby evita la dependencia de una tecnología externa, constituyéndose como una alternativa en Java capaz de trabajar con bases de datos embebidas, sin tener la necesidad de instalar

un servidor de base de datos externo a la JVM, garantizando un único punto de acceso desde el sistema. Además, como no existen accesos concurrentes debido a que solo se cuenta con una estación de trabajo, no es necesario proveer su gestión como lo posibilita PostgreSql (The Apache Software Foundation, 2014).

1.5 Metodología de desarrollo de software

A continuación se definirán los procedimientos a emplear para el correcto desarrollo del sistema a implementar, los cuales serán elegidos a partir de la Metodología de Desarrollo de Software que se seleccione.

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. Estas se clasifican en dos grupos:

- Metodologías orientadas al control de los procesos: Establecen rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Estas metodologías son llamadas Metodologías Pesadas. (Isasias Carrillo Perez, Rodrigo Perez Gonzalez, Aureliano David Rdriguez Martin, 2008) Este enfoque pesado implica una importante sobrecarga de trabajo en cuanto a la planificación, diseño y documentación del sistema. Esto es justificado cuando se tiene que coordinar el trabajo de múltiples equipos de desarrollo, donde existe determinada criticidad para el sistema y cuando muchas personas están involucradas en el mantenimiento del software durante su vida. (Ian Somerville, 2005)

- **Metodologías orientadas a la interactuación con el cliente y el desarrollo incremental del software**: Estas proponen mostrar versiones principalmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando. Son llamadas Metodologías ligeras o ágiles (Isasias Carrillo Perez, Rodrigo Perez Gonzalez, Aureliano David Rdriguez Martin, 2008). Permiten a los equipos de desarrollo centrarse en el software mismo en vez de en su diseño y documentación. Las metodologías ágiles dependen de un enfoque iterativo para la especificación, desarrollo y entrega del software. Fueron diseñadas principalmente para apoyar el desarrollo de aplicaciones de negocio, donde los requerimientos del sistema normalmente cambian con rapidez durante el proceso. Están pensadas para entregar software funcional de forma más rápida a los clientes, los que pueden proponer que se incluyan en iteraciones posteriores del sistema nuevos requerimientos o cambios en los mismos. (Ian Somerville, 2005)

De acuerdo a las características de las metodologías ágiles en correspondencia con la pequeña cantidad de integrantes del equipo de desarrollo, el aprovechamiento de la disponibilidad del cliente, el poco tiempo disponible para el desarrollo y la poca documentación que se requiere generar, se impone escoger una de las metodologías ágiles existentes. Por su semejanza con estas características, luego

de realizar una investigación, la mejor candidata para la selección resultó ser XP o Programación Extrema por la descripción plasmada a continuación.

1.5.1 XP o Programación Extrema

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (Jose H. Canos, 2003)

Los defensores de XP consideran que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista a definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios de estos. Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto y aplicarlo de manera dinámica durante el ciclo de vida del software. (Roger S. Pressman, 1997)

XP posee las siguientes características que desde la perspectiva del entorno del desarrollo de la solución propuesta pueden ser defendidas:

- La simplicidad del diseño para agilizar el desarrollo y facilitar el mantenimiento:
- Definir un diseño simple es conveniente al objetivo que persigue la investigación, debido a que se pretende mejorar la mantenibilidad del sistema existente.
- La retroalimentación dada por la integración del cliente al equipo de desarrollo :
- Aprovechando la localización de la librería en la universidad, se puede sostener una continua retroalimentación ante la sucesión de cambios en los requisitos o la decisión de las características prioritarias, mostrar resultados en ciclos muy cortos, así como informar sobre el estado de salud del código mediante las pruebas, participando este cliente en el equipo de desarrollo como un miembro más.
- Generación de poca documentación.

Dado que el tiempo disponible para el desarrollo es corto y los procesos de negocio de la librería no son complejos, es conveniente generar la menor cantidad de documentos posible.

En contraste con el uso de XP, se utilizarán herramientas para la implementación de la solución, cuya selección se describe y justifica en el siguiente epígrafe.

1.6 Herramientas y tecnologías de desarrollo

Las herramientas y lenguajes que se describen a continuación, son seleccionadas de acuerdo a las necesidades de la implementación de la solución.

1.6.1 BPMN

La Notación para el Modelado de Procesos de Negocio o Business Process Model and Notation (BPMN) es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. (OMG, 2011) El principal objetivo de BPMN es facilitar una notación estándar que sea fácilmente legible, tiene la meta de servir como lenguaje común entre el diseño de los procesos de negocio y su implementación. Los procesos que serán informatizados son modelados utilizando esta notación para proporcionar un mejor entendimiento de los mismos.

1.6.2 Herramienta CASE

Las Herramientas para la Ingeniería de Software Asistida por Computadora (CASE) son un conjunto de programas para facilitar el trabajo de analistas, ingenieros de software y programadores, durante el ciclo de vida del desarrollo de un Software. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar el diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras. (Krishnamurthy, 2011)

Una de estas herramientas es el **Visual Paradigm**, herramienta de modelado profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. La misma facilita una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Es importante aclarar, que aunque la metodología XP no exige generar como artefactos diagramas de clase o de componentes del diseño como sucede en esta investigación, estos se diseñaron con el objetivo de reflejar conceptos importantes del negocio y el diseño, así como las relaciones entre sus componentes Algunas características de Visual Paradigm que se tuvieron en cuenta para su selección fueron:

- Entorno de creación de diagramas para UML 2.0.
- Exporta los diagramas y diseños en formatos: JPEG, PDF etc.
- Es un software multiplataforma soportado en Java para sistemas operativos como Windows y Linux.
- Convierte los diagramas de entidad-relación a tablas de base de datos.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa. (Suarez, 2012)

1.6.3 Selección del lenguaje de programación

Los lenguajes de programación son idiomas artificiales diseñados para expresar cálculos y procesos que serán llevados a cabo por ordenadores. Un lenguaje de programación está formado por un conjunto de palabras reservadas, símbolos y reglas sintácticas y semánticas que definen su estructura y el

significado de sus elementos y expresiones. El proceso de programación consiste en la escritura, compilación y verificación del código fuente de un programa. (Guevara de. Desc., 2012)

JAVA

JAVA es un lenguaje muy valorado porque los programas implementados con su utilización se pueden ejecutar en diversas plataformas con sistemas operativos como Windows, Mac OS, Linux o Solaris. Para conseguir la portabilidad de los programas Java se utiliza un entorno de ejecución para los programas compilados. Este entorno se denomina Java Runtime Environment (JRE). Es gratuito y está disponible para los principales sistemas operativos. (Guevara de. Desc., 2012)

Este lenguaje de programación se selecciona para el desarrollo de la aplicación teniendo en cuenta que en él se pueden desarrollar tanto aplicaciones web como de escritorios siendo el caso. Posee restricciones que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora. Además como el código compilado de Java es interpretado, un programa puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java, es decir la Máquina Virtual Java (JVM), haciéndolo un lenguaje portable. Java logra ser un lenguaje independiente de una arquitectura computacional definida, porque cuando se compila un programa, el código resultante es interpretado por diferentes computadoras de igual manera además, puede ejecutar diferentes líneas de código al mismo tiempo. Otras de las ventajas que brinda es que no requiere que se compilen todas las clases de un programa para que este funcione, por lo tanto es dinámico y robusto (P. J. Deitel, 2006).

1.6.4 Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (IDE) es un programa informático compuesto por un conjunto de herramientas, utilizadas por los programadores para desarrollar código. Está constituido por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien para varios. (Community, 2010)

Netbeans 8.0

Netbeans es un IDE que se ejecuta en varias plataformas incluyendo Windows y Linux. Dispone de soporte para crear interfaces gráficas de forma visual, control de versiones y sus funcionalidades son ampliables mediante la instalación de paquetes. Es ideado mayormente para la programación en el lenguaje Java sin contar que es un producto libre y gratuito sin restricciones de uso, alcanzando un gran éxito como proyecto de código abierto. Puede obtener todas las herramientas que necesite para crear aplicaciones profesionales para el escritorio, la empresa, la web y equipos móviles con los lenguajes Java, C/C++, y Ruby, (Community, 2010).

Además de ser un IDE de código abierto, tener un potente motor para la conexión a base de datos, el resaltado de líneas, el completamiento de código y la adaptabilidad hacia los estándares de codificación definidos son de más fácil uso; implica también la actualización del JDK.

NetBeans IDE 8.0 proporciona analizadores de código y editores para trabajar con las últimas tecnologías Java 8 - Java SE 8, Java SE Embedded 8 y Java ME Embedded 8 (Oracle, 2015).

Otras de las importantes prestaciones que ofrece esta versión del IDE son:

- Soporte al JDK 8: Provee herramientas y mejores editores para trabajar con perfiles, Lambdas y Streams.
- Soporte a Java SE Embedded: Permite implementar, depurar o perfilar aplicaciones Java SE en un dispositivo incrustado, directamente desde el Netbeans; condición favorable para la utilización de Apache Derby como gestor de la base de datos del sistema a implementar.
- Mejoras del editor de Java:
    - Nuevos Java Hints.
    - Se presenta el JavaDoc como una información de herramientas.
    - Mejoras en el cambio instantáneo de nombres y en el completamiento de código.
    - Integración mejorada con el JavaFX Scene Builder, cuya utilización es requerida para el diseño de las interfaces graficas de usuario.

Conclusiones del capítulo

- El estudio de los atributos de usabilidad y mantenibilidad permitió obtener los conocimientos teóricos necesarios a tener en cuenta en la implementación de la solución. para relacionar los aspectos de la problemática referidos a los mismos.
- El análisis de los procesos de gestión de la información y el control de libros en la Librería de la Universidad de las Ciencias Informáticas permitió detectar deficiencias funcionales y tecnológicas de la solución informática existente, lo que evidenció la necesidad de proponer una solución que implique un cambio de tecnologías y herramientas para su implementación, orientado a incrementar la usabilidad y la mantenibilidad del sistema.
- La caracterización del entorno del proceso de desarrollo de la solución propuesta, permitió identificar la necesidad de utilizar una de la metodologías agiles existentes para guiar el proceso de desarrollo del software.
- Para el desarrollo de la propuesta de solución se seleccionó:
  - VisualParadingm 9.1 para el diseño de los diagramas necesarios para el modelado del negocio que se informatiza.
  - Java SE 8.3.1 con el JDK de la misma versión, el cual brinda soporte a la utilización de Java FX Scene Builder para la implementación de las interfaces graficas de usuario; y al Gestor de Base de Datos Apache Derby en su versión 8.0.

CAPÍTULO 2: PLANIFICACION, DISEÑO E IMPLEMENTACION

Introducción

El presente capítulo describe el funcionamiento general del sistema. Se presentan los requisitos de la solución propuesta detallados a partir de la confección de las Historias de Usuario en el caso de los funcionales. Se obtienen los artefactos generados por la metodología XP en cada una de sus fases y algunos no exigidos pero si útiles en la descripción de conceptos fundamentales de la implementación de la solución propuesta.

2.1 Descripción del sistema

La solución propuesta es una versión actualizada del Sistema para la Gestión de información y Control de Libros v 1.0 para la librería de la Universidad de las Ciencias Informáticas, donde las principales funcionalidades encargadas de la gestión y control de literatura y la gestión de reportes se mantienen, así mismo sucede con el tratamiento de errores, la gestión del nivel de acceso de los usuarios al sistema y a la información que se manipula en el mismo. Permite la posibilidad de auditar los movimientos que se realicen en un periodo de tiempo determinado; acción que le facilita máximo directivo de la entidad tener un control sistemático dentro de la misma, desde el interior de los procesos donde la literatura está involucrada.

Para ello se establecieron los siguientes roles:

Administrador (a): Único usuario que posee todos los privilegios para acceder a toda la información gestionada en el sistema.

Almacenero: Posee acceso a la actualización de la información del inventario de almacén y a la generación de los informes de recepción, los vales de salida y de devolución.

Económica: actualiza el inventario de salón, visualiza la información del inventario del almacén y tiene los permisos para generar los reportes diarios de ventas.

Para implementar el sistema descrito se emplean como guía las fases que propone la metodología XP descritas a continuación.

2.2 Planificación

2.2.1 Requisitos funcionales

Como fase inicial de la Programación Extrema, en la planificación los clientes plantean a grandes rasgos las historias de usuario (HU) que son de interés para la primera entrega del producto. Estas describen los requisitos funcionales los cuales constituyen las funciones que el sistema debe cumplir para satisfacer las necesidades del cliente. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. (Jose H. Canos, 2003) Durante esta fase debe existir un constante diálogo entre el equipo de desarrollo y el usuario final, para poder determinar qué es lo que verdaderamente se quiere hacer y el orden en que deben hacerse las funcionalidades que sean definidas, estableciendo un cronograma de entrega.

Definir las historias de usuario es el primer paso. Estas tienen una prioridad establecida por el usuario con la cual el equipo de desarrollo realiza la valoración de cuánto tiempo puede demorar la realización de cada una de ellas, para poder acordar las funcionalidades que serán entregadas en cada iteración y publicar un plan de entrega donde se reflejan las fechas de liberación de las iteraciones establecidas. (Castillo Oswaldo)

Como resultado de la definición de las Historias de Usuario (ver Anexos), se obtuvo la descripción de los requisitos funcionales que se relacionan a continuación, siendo **Gestionar eventos** y **Visualizar eventos** los incorporados a la nueva versión del sistema que pretende solucionar el problema de la investigación. Al resto de las funcionalidades solo se le implementaran mejoras con el objetivo de mejorar su usabilidad y mantenibilidad:

- Autenticar usuario
- Gestionar usuario
- Gestionar libro
- Gestionar Informe de Recepción
- Gestionar Resumen de Ventas
- Gestionar Vale
- Generar Tarjeta de Estiba
- Generar Reporte Diario de Operaciones
- Visualizar informes
- Gestionar eventos
- Visualizar eventos

A continuación se muestra la definición para la Historia de Usuario **Gestionar eventos**, encargada de describir la funcionalidad responsable de la auditoria de los movimientos realizados dentro del sistema. El resto se puede consultar en el anexo No.1.

Historia de Usuario	
Número: 10	Usuario: Todos los usuarios
Nombre de historia: Gestionar Evento	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados: 2 semana	Iteración asignada: 1
Programador responsable: Dayana Donatien Morejón	
Descripción: •Registrar evento: el usuario selecciona la opción registrar para la aceptación de registro de documentos. El sistema brinda la posibilidad de registrar del evento el lugar, fecha, hora y usuario protagonista.	

Tabla 1: Historia de Usuario Gestionar eventos.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son las características que debe poseer el sistema para que sea un producto de fácil uso, rápido y sobre todo con la calidad requerida.

Apariencia o interfaz externa

La interfaz de la aplicación debe estar diseñada con un estilo común garantizando que exista uniformidad, manteniendo de forma única la manera de presentar los contenidos, facilitando la realización de cambios en ellas. Se deben emplear colores sencillos que transmitan una sensación de seriedad y confianza.

Usabilidad

El diseño e implementación de las interfaces graficas de usuario debe estar orientada a minimizar el acoplamiento entre las clases definidas, en aras de que:

- Se garantice el logro de la satisfacción de facilidad de aprendizaje y la comprensión de la interacción con el sistema, partiendo de que el esfuerzo necesario para aprender a operar con el mismo, preparar los datos de entrada e interpretar sus salidas sea el mínimo.

Mantenibilidad

Con el objetivo de garantizar que el sistema sea posible de corregir si se encuentra un error, de adaptar si su entorno cambia, o mejorar si el cliente desea un cambio de requisitos con un bajo nivel de dificultad; debe tenerse en cuenta que: .

- Durante su desarrollo, la fuerza de relación entre las clases componentes tenga el nivel más bajo posible, así como la complejidad de implementación; esta última regida por la cohesión de las clases.

Soporte

Debe brindar una dirección de contacto para poder responder a los problemas de los usuarios y otras vías de comunicación con el equipo de realización del software. Además se requiere que el producto reciba mantenimiento ante cualquier cambio que ocurra. El sistema debe ser de fácil instalación y proveer una ayuda para que sea consultada en caso de necesidad.

Portabilidad

Para la implementación del sistema se debe emplear herramientas de programación y gestión de bases de datos multiplataforma, garantizando que el mismo pueda ser utilizado en diferentes sistemas operativos como Windows y Linux.

Seguridad

El sistema debe ser capaz de controlar los diferentes accesos y privilegios de los usuarios, además de identificar al usuario antes de que pueda realizar cualquier acción sobre el sistema.

Auditabilidad

El sistema debe permitir auditar los movimientos que se realicen con la información existente en un periodo de tiempo determinado. Con este objetivo se propone aplicar un mecanismo que dentro de la misma implementación garantice esta propiedad para el sistema. En este caso se ajusta muy bien la utilización del logging o generación de los ficheros log, que es quizás la herramienta más potente para auditar el sistema. El logging permite que un sistema operativo o aplicación grabe eventos mientras ocurren y los guarda para un examen posterior. Es un componente esencial de cualquier sistema operativo, al cual se le debe prestar mucha atención. En cuanto a seguridad permite mantener un registro de las acciones dañinas de un atacante. Estos registros se guardan en el directorio /var/log.

Software

Para el funcionamiento del sistema será necesario el Sistema Operativo Windows XP o superior, Linux o Unix, además la máquina virtual de JAVA (JVM) en su versión 1.8.31.

Hardware.

Se necesitan como requisitos mínimos una PC con procesador Pentium 4 o superior, con 1GB de RAM o superior y una impresora.

2.2.3 Validación de requisitos

Una vez descritos a grandes rasgos los requisitos funcionales del sistema mediante las historias de usuario y definidos los requisitos no funcionales, es prudente examinar sus especificaciones para asegurar que han sido establecidos sin ambigüedad, inconsistencias u omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. Esta tarea se realiza mediante métricas de la calidad de especificación de requisitos.

Métricas

El Glosario de Estándares del IEEE define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado. (Roger S. Pressman, 1997)

Especificidad de los requisitos

A partir de una reunión realizada con el cliente con el objetivo de concretar interpretaciones comunes de los requisitos, se aplicó la métrica de la calidad de especificación .El objetivo es cuantificar la especificidad o falta de ambigüedad en la definición de los requisitos. Para su cálculo deben contarse los requisitos que tuvieron igual interpretación por los revisores y compararlos con el total de requisitos definidos. La especificidad de los requisitos se calcula como:

$$Q_1 = \frac{n_{ui}}{n_r}$$

Donde:

Q1: Especificidad de los requisitos.

Nr: Total de requisitos definidos.

Nui: Total de requisitos para los que los revisores tuvieron interpretaciones idénticas.

El valor de esta métrica debe estar siempre entre 0 y 1. Mientras más cerca de 1 esté el valor de especificidad mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos.

La siguiente tabla evidencia la interpretación obtenida por los revisores de los requisitos funcionales y no funcionales.

Atributo de calidad	Tipo de requisito	Interpretaciones	
		Iguales	Desiguales
Especificidad	Funcional	9	0
	No funcional	8	0
	Total	17	0

Tabla 2: Interpretación de los revisores de la especificación de requisitos.

En correspondencia con los datos recopilados se realizaron los cálculos pertinentes y se obtuvo como resultado una especificidad Q1 = 1 lo que significa que los requisitos especificados no son ambiguos.

Grado de validación de los requisitos

Los requisitos deben ser posibles de validar. En consenso del equipo de desarrollo se contrasta lo que desea el cliente con la posibilidad real de implementarlo. El grado de validación de los requisitos mide la corrección en la definición de los requisitos. (Roger S. Pressman, 1997)

Este valor se calcula como:

$$Q_3 = \frac{n_c}{(n_c + n_{nv})}$$

Donde:

Q3: Grado de validación de requisitos.

Nc: Total de requisitos validados correctamente.

Nnv: Total de requisitos no validados.

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo es el más cercano a 1 e indica un alto nivel de corrección en la definición de los requisitos.

Una vez aplicada esta métrica se obtuvieron los siguientes resultados:

Q3 = 17 / (17 + 0) = 17 / 17= **1**

Se evidencia entonces que la definición de los requisitos es óptima.

Estabilidad de los requisitos:

El objetivo de esta métrica es medir cuán estables son los requisitos. Se considera que los requisitos son estables cuando no existen adiciones o supresiones en ellos que impliquen modificaciones en las funcionalidades principales de la aplicación. La estabilidad de los requisitos se calcula como: (Suarez, 2012)

$$ETR = \left[\frac{RT - RM}{RT} \right] * 100$$

Donde:

RT: Total de requisitos definidos.

RM: Cantidad de requisitos modificados.

Esta métrica ofrece valores entre 0 y 100. El mejor valor de ETR es el más cercano a 100 porque mostrará que no se están realizando cambios sobre los requisitos, son estables y por tanto, es confiable trabajar el análisis y diseño sobre ellos.

Una vez aplicada esta métrica para los 17 requisitos definidos se obtuvo el siguiente resultado:

ETR = [(17 – 1) / 17] * 100 = 94.

Con este resultado se puede concluir que la estabilidad para los requisitos definidos es aceptable, pues su valor es bien cercano a 100.

2.2.4 Estimación de esfuerzo por historia de usuarios

Las Historias de Usuarios proporcionan suficientes detalles para hacer una estimación razonable de cuánto tiempo le tomará a los programadores implementarlas (Don Wells, 2013). El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. (Jose H. Canos, 2003).

La Tabla 3 relaciona a cada uno de los requisitos funcionales definidos, con la respectiva estimación del esfuerzo que requiere su implementación determinado por la cantidad de semanas. Se obtiene como resultado que para desarrollar las funcionalidades, serán necesarias 18 semanas aproximadamente,

Historia de Usuario	**Punto de estimación (semanas)**
Autenticar usuario	1
Gestionar usuario	1
Gestionar libro	1
Gestionar Informe de recepción	2
Gestionar Resumen de Ventas	2
Gestionar vale	2
Generar Reporte diario de operaciones	1

Generar Tarjeta de Estiba	2
Visualizar informes	2
Gestionar evento	2
Visualizar eventos	2
Total	18

Tabla 3: Estimación de esfuerzo por historia de usuario.

Plan de duración de las iteraciones

En correspondencia con el tiempo que implica la realización de cada historia de usuario, se decide dividir la implementación del sistema en diferentes iteraciones una vez consultada la opinión del cliente y definidas las principales funcionalidades; por tal motivo se establece el plan de duración de las iteraciones. Este plan define exactamente cuáles historias de usuario serán implementadas para cada iteración. Se decide realizar el sistema en dos iteraciones y debido a que existe un único equipo de desarrollo se elaboró un solo plan de iteraciones, el cual se muestra a continuación (ver Tabla 4):

Iteración	**Historia de usuario a implementar**	**Duración de la iteración**
1	Autenticar usuario Gestionar usuario Gestionar libro Gestionar Informe de recepción Gestionar vale Gestionar evento	9 semanas
2	Gestionar Resumen de Ventas Generar Tarjeta de Estiba Generar Reporte diario de operaciones Visualizar informes Visualizar Eventos	9 semanas

Tabla 4: Plan de duración de las iteraciones.

Plan de entrega

El plan de entrega especifica qué historias de usuario se van a implementar para cada iteración del sistema y las fechas para las liberaciones de las mismas (Don Wells, 2013).

Se presenta a continuación el plan de entrega, elaborado con las fechas en que serán liberadas las historias de usuario.

Iteración	**Historia de usuario a implementar**	**Fecha de entrega**

1	Autenticar Usuario Gestionar Usuario Gestionar Libro Gestionar informe de recepción Gestionar vale Gestionar Evento	15/04/2015
2	Gestionar Resumen de Ventas Generar Tarjeta de Estiba Generar reporte diario de operaciones Visualizar informes Visualizar Eventos	17/06/2015

Tabla 5: Plan de entrega

2.3 Diseño

La metodología XP defiende la confección de un diseño simple cuyo tiempo de acabado generalmente es menor que el de uno complejo. Sugiere hacer siempre las cosas más simples que podrían funcionar en el futuro teniendo en cuenta que es más económico reemplazar el código complejo antes de emplear tiempo innecesario en él. Este diseño debe tener una estructura que sea entendible a nuevos integrantes del proyecto que puedan contribuir rápidamente al mismo, y que nombre clases y métodos consistentes (Don Wells, 2013). Propone usar glosarios de términos si es necesario y una correcta especificación de los nombres de métodos y clases; pues esto ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código.

Modelo de datos

El modelo de datos como artefacto que se genera en la fase de diseño, permite describir las estructuras de los datos y la forma en que se relacionan. Incluye además las restricciones de integridad, lo cual permite definir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí. Para el sistema a desarrollar se obtuvo un modelo que consta de un total de 19 tablas, de ellas 7 nomencladores y las 12 restantes para el almacenamiento de la información manipulada por el Sistema para la Gestión de información y Control de Libros v 2.0. Para ver el modelo de datos consultar el anexo No.4.

Diagrama de clases

Un diagrama de clases de diseño representa las especificaciones de las clases de una aplicación. En él se contemplan las clases, sus atributos y funcionalidades, la relaciones entre ellas, navegabilidad, dependencia, entre otros elementos. Cada clase describe algún elemento del dominio del problema, con enfoque en los aspectos del problema visibles para el usuario o el cliente. Aunque la metodología XP no sugiere la realización de un diagrama de clases, este se realizó en aras de definir cuáles son las

clases fundamentales para el desarrollo de la aplicación. Teniendo en cuenta que lo más importante que se realiza en la librería son las operaciones con los libros, controladas mediante los documentos correspondientes a cada una; se definen que las clases que rigen el desarrollo de la aplicación propuesta, son las que se muestran en el siguiente diagrama diseñado para modelar la relación entre las clases persistentes solamente (ver Ilustración 6).

.

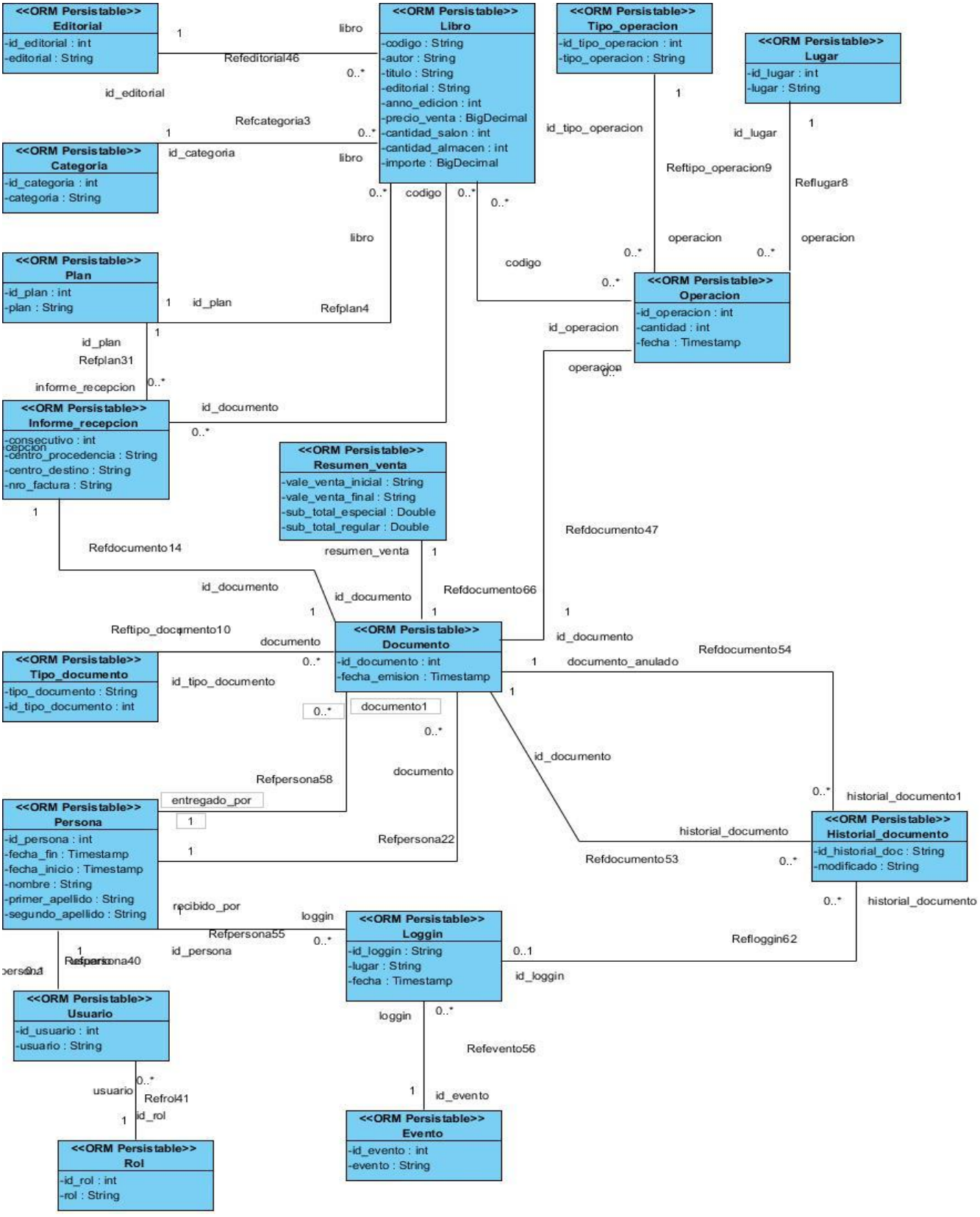


Ilustración 6: Diagrama de clases persistentes.

Patrón Arquitectónico

Los patrones de arquitectura expresan un esquema organizativo estructural fundamental para sistemas de software, especificando un conjunto de pequeños sistemas interrelacionados con sus responsabilidades y recomendaciones para los distintos componentes que conforman el sistema. Para el desarrollo de la aplicación se emplea el patrón Modelo-Vista-Controlador (MVC), que permite realizar la programación multicapa, separando en tres componentes distintos: los datos de una aplicación, la interfaz del usuario y la lógica de control.

• Modelo: representa la información con la que trabaja la aplicación, o sea, su lógica de negocio.

• Vista: se encuentran las clases que muestran al usuario la información existente en el modelo.

• Controlador: el controlador es el encargado de aislar al modelo y a la vista de los detalles del protocolo usado para las peticiones. (Alpaev, 2006)

• Clases objeto: se encuentran las clases correspondientes a los objetos persistentes que son utilizados indistintamente en cada una de las capas restantes.

Ilustración 7: Patrón arquitectónico Modelo – Vista - Controlador.

Patrones de diseño

Un patrón de diseño constituye una pareja de problema - solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas. (Craig Larman, 1999)

Estos se dividen en dos grupos:

- **General Responsibility Asignment Software Patterns**, GRASP por sus siglas en inglés. Éstos representan los principios básicos de la asignación de responsabilidades a objetos.
- **Gang of Four**(o GoF por sus siglas en inglés) que se clasifican en tres categorías basadas en su propósito (Kord, 2011):

- **Creacionales**: encargados de abstraer el proceso de creación de instancias y ocultar los detalles de cómo los objetos son creados o inicializados.
- **Estructurales**: estos patrones se ocupan de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades.
- **Comportamiento**: relacionados con los algoritmos y la asignación de responsabilidades entre los objetos. Son utilizados para organizar, manejar y combinar comportamientos

A continuación se reflejan los patrones de diseño que son aplicados en la implementación del sistema:

GRASP

- **Experto**

Este patrón se evidencia en el sistema a partir de la definición de clases responsables de ejecutar funcionalidades específicas, debido a que contienen la información necesaria para su cumplimiento. Es beneficioso la utilización del patrón Experto porque se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, favoreciendo el hecho de implementar un sistema más robusto y de fácil mantenimiento. Además el comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases más sencillas y cohesivas, fáciles de comprender y mantener, brindando soporte a una alta cohesión.

Por ejemplo en el sistema la clase **LibroObjeto** es la encargada de generar las tarjetas de estiba porque es la que contiene la información referente a los libros involucrados. La siguiente ilustración visualiza lo antes expuesto.

```
 * @author developer
 */
public class LibroObjeto extends WrapperEntity2Object{
    private final Libro l;
    private int cantidadTraslado;


    public TarjetaEstiba obtenerTarjetaEstiba(ENUM_LUGAR el) throws NoSuchMethodException,
            IllegalAccessException, IllegalArgumentException, InvocationTargetException{
        TarjetaEstiba te = OperacionTableAccess.getInstance().obtenerOperacionesPorLibro(l, el);
        return te;
    }
```

Ilustración 8: Patrón Experto.

- **Creador**

Se definen clases en el sistema encargadas de la creación de objetos. El propósito fundamental consiste en encontrar un Creador que debe conectarse con el objeto producido en cualquier evento, por ejemplo, asignar la responsabilidad de que una clase B cree un objeto de la clase A, teniendo en

cuenta que en el momento de crear objetos tienen importancia las características de la clase. (Craig Larman, 1999) En el sistema este patrón se aplica principalmente en las clases Manager o Controladoras, que son las encargadas de crear instancias de las clases del modelo.

Utilizar el patrón Creador brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase Creador, debido a las asociaciones actuales que conllevan a elegirla como el parámetro adecuado.

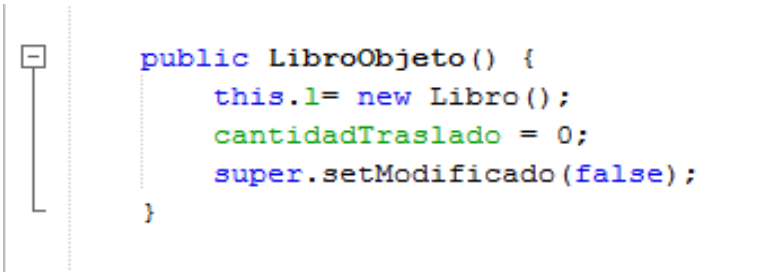


Ilustración 9: Patrón Creador.

- **Bajo acoplamiento**

Con el objetivo de dar soporte en la disminución de la medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas, se tiene presente este patrón durante la implementación. Independientemente de que con los descritos anteriormente se condiciona de forma aceptable, se le da vital importancia a este patrón durante todo el desarrollo. Una clase con bajo o débil acoplamiento no depende de muchas otras. Este patrón asigna la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. En la confección de las tarjetas CRC se muestra concretamente la relación entre las clases y la función específica de cada una observándose también las escasas dependencias entre las mismas. Mantener un acoplamiento bajo, tributa a que la futura mantenibilidad esté garantizada en gran medida.

- **Alta cohesión**

La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. El empleo de otros patrones como el Creador y el Experto lo favorece, del mismo modo que lo hace la confección de las tarjetas CRC que reflejan que las clases tienen responsabilidades moderadas en un área funcional y colaboran con las otras para llevar a cabo las tareas.

- **Controlador**

La utilización de este patrón en el diseño del sistema se observa, en la definición de objetos de interfaz no destinada al usuario que se encargan de manejar eventos del sistema, evidenciando además los métodos de sus operaciones. Estos eventos son generados por actores externos, obteniendo como

respuesta operaciones del sistema. Los objetos Controladores definidos delegan en otros el trabajo que se necesita hacer; coordinando o controlando la actividad. (Craig Larman, 1999)

En el sistema, las clases Controladoras son las encargadas de establecer la comunicación entre la vista y el modelo, cuya responsabilidad es contener un conjunto de funcionalidades que permite llevar un control de la lógica del negocio.

```
public void RegistrarLibros() throws Exception {
    if (librosNuevos.isEmpty()) {
        throw new Exception("No se ha registrado ningun libro aun.");
    } else {
        try {
            for (LibroObjeto libroObjeto : librosNuevos) {
                libroObjeto.save();
            }
        } catch (PersistenceException pe) {
            throw new Exception(pe.getMessage());
        }
    }

    librosNuevos = new LinkedList<LibroObjeto>();
}
```

Ilustración 10: Patrón Controlador.

GoF

- **Singleton**

  Con la intención de garantizar que determinadas clases tengan una sola instancia y proporcionar un acceso global a ella, se utilizó este patrón. De esta forma se restringe la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Por ejemplo en la aplicación se puede ver que en la clase UsuarioManager el constructor se declara privado para garantizar que no se creen instancias innecesarias de la clase, para tener acceso a esta clase y sus funcionalidades se declaró el método público getInstance. A continuación se muestra una imagen donde se aplica dicho patrón.

```
public class UsuarioManager {
    private static UsuarioManager um;

    private UsuarioManager() {
    }

    public static UsuarioManager getInstance(){
        if(um == null){
            um = new UsuarioManager();
        }
        return um;
    }
```

Ilustración 11: Patrón Singleton.

- **Observer**

  En aras de mejorar, desde el punto de vista de la implementación, la gestión de la presentación y actualización de los datos que fluyen en los procesos que informatiza el sistema, se decide a aplicar de este patrón. Este permite que varios objetos observadores de eventos sean notificados cuando los objetos sujetos a estos cambian. Cada observador se registra con el sujeto mediante un oyente, y cuando se produce un cambio este se encarga de notificar al observador. Tal es el caso de la clase SeleccionarLibrosController donde la funcionalidad changed es la encargada de estar pendiente a los cambios del lugar para el traslado de los libros (ver Ilustración 12).Esto implica que la complejidad de implementación disminuya en el momento de actualizar los datos antes de ser mostrados al usuario.

```
        @Override
        public void changed(ObservableValue<? extends ENUM_LUGAR> ov, ENUM_LUGAR t, ENUM_LUGAR t1) {
            ENUM_LUGAR l = ENUM_LUGAR.ALMACEN;
            if (t1.equals(ENUM_LUGAR.SALON)) {
                tc_existenciaAlmacen.setVisible(true);
                tc_existenciaSalon.setVisible(false);
            } else if (t1.equals(ENUM_LUGAR.ALMACEN)) {
                tc_existenciaSalon.setVisible(true);
                tc_existenciaAlmacen.setVisible(false);
                l = ENUM_LUGAR.SALON;
            }
            SortedList<LibroObjeto> filtereableCollection = UIUtiles.initFiltereableCollection(FXCollections.observableArrayList(
            filtereableCollection.comparatorProperty().bind(tblLibrosTraslado.comparatorProperty());
            tblLibrosTraslado.setItems(filtereableCollection);
        }
    });
```

Ilustración 12: Patrón Observer.

- **Adapter**

  En determinados objetos del diseño del sistema, es necesario permitir adaptar lo que estos ofrecen para que otros puedan hacer uso de ellos. Tal es el caso de LibroManager, la cual necesita obtener información de la clase persistente Libro. Dado que la clase LibroObjeto es la experta en la información referente a los libros, esta es utilizada para responder a estas peticiones, adaptando de esta forma lo que ofrece la clase Libro para ser utilizado por LibroManager.

  La Ilustración 13 visualiza para la funcionalidad RegistrarLibros de la clase Controladora LibroManager lo antes expuesto.

```
public void RegistrarLibros() throws Exception {
    if (librosNuevos.isEmpty()) {
        throw new Exception("No se ha registrado ningun libro aun.");
    } else {
        try {
            for (LibroObjeto libroObjeto : librosNuevos) {
                libroObjeto.save();
            }
        } catch (PersistenceException pe) {
            throw new Exception(pe.getMessage());
        }
    }

    librosNuevos = new LinkedList<LibroObjeto>();
}
```

```
 * @author developer
 */
public class LibroObjeto extends WrapperEntity2Object{
    private final Libro l;
    private int cantidadTraslado;
```

Ilustración 13: Patrón Adapter.

- **Factory**

  Con la utilización de este patrón en el sistema se definen clases encargadas de la creación de objetos específicos, tributando a que disminuya la complejidad de implementación desde esta arista. Tal es el caso de la clase MessageFactory, la cual permite que cada vez que se necesite crear un mensaje determinado solo tenga que proveer los parámetros necesarios para ello en dependencia del tipo que sea. La siguiente ilustración visualiza el procedimiento para crear un mensaje.

```
public class MessageFactory {

    private static VBox msgStructure = null;
    private static FlowPane buttonsArea = null;
    private static Label msgArea = null;
    private static Stage msgStage = null;
    private static Scene msgScene = null;
    private static Tooltip ttp;
    private static final String msgCierreTooltip = "\nPresione ESC para cerrar este mensaje.";
    private static ObjectProperty<ENUM_MSG_EXITS> exit;
    private static EventHandler<ActionEvent> closeMsgOK = null;

    private static EventHandler<ActionEvent> closeMsgYES = null;

    private static EventHandler<ActionEvent> closeMsgNO = null;

    private static EventHandler<ActionEvent> closeMsgCANCEL = null;

    private static void initMsgStructure() throws IOException {
        if (msgStructure == null) {
            Parent msg = FXMLLocator.getInstance().getFXML("Mensajes/messagePrototype.fxml");
            msgStructure = (VBox) msg;
            ObservableList<Node> children = msgStructure.getChildren();
            children.stream().forEach((node) -> {
                if (node instanceof FlowPane) {
                    buttonsArea = (FlowPane) node;
                    buttonsArea.getChildren().clear();
                } else if (node instanceof AnchorPane) {
                    AnchorPane ap = (AnchorPane) node;
                    msgArea = (Label) ap.getChildren().get(0);
                }
            });
```

Ilustración 14: Patrón Factory.

- **Mediator**

Se utiliza este patrón con el objetivo de apoyar la coordinación entre los objetos de la interfaces gráficas de usuario y por consiguiente disminuir el acoplamiento entre las mismas, debido a que permite obtener en una clase específica, el reporte de los cambios de estado de estos objetos. Esto provee la posibilidad de tener un punto de intercambio, donde cada ventana en su interfaz gráfica de usuario sólo tiene que interactuar con dicho punto, y no necesita entender todas las funcionalidades para comunicarse con el resto de las ventanas como sucede en la versión 1.0.De esta forma se aumenta la reutilización garantizando que el futuro mantenimiento del sistema se vea favorecido.

En el siguiente diagrama de clases se pretende reflejar la utilización de este patrón en el sistema mediante la representación de la relación de uso entre las clases AdicionarLibroController e InformeRecepcionController; donde la clase Mediator juega el papel de intermediario.

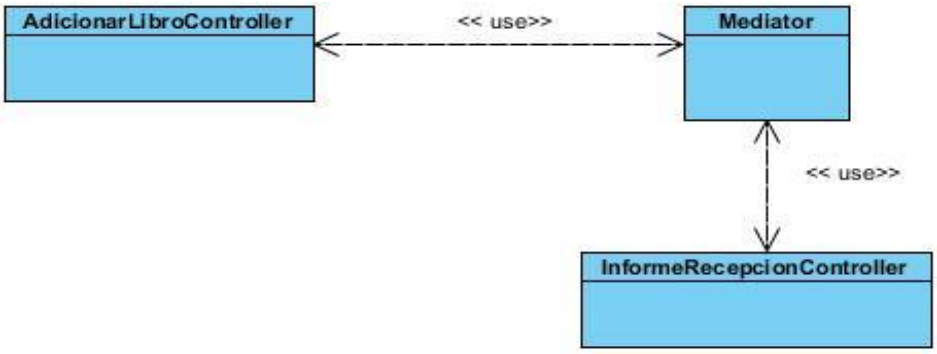


Ilustración 15: Patrón Mediator.

Tarjetas CRC

En esta fase XP propone el uso de las Tarjetas de Clases, Responsabilidades y Colaboradores, C.R.C por sus siglas en español, que permiten al programador centrarse y apreciar el desarrollo orientado a objetos. Utilizando las tarjetas C.R.C. se determina cuáles son las funcionalidades que deben cumplir las clases y su interrelación en la implementación de dichas funcionalidades (Castillo Oswaldo).

Se define una por cada clase, sobre las cuales se abrevian las responsabilidades de la misma y se anota una lista de los objetos con los que colaboran para desempeñarlas. Suelen elaborarse en una sesión de grupos pequeños donde los participantes representan papeles de las diversas clases. Cada grupo tiene las tarjetas CRC correspondientes a las clases cuyo papel está desempeñando (Craig Larman, 1999).

Las tarjetas CRC representan objetos, la clase a la que pertenece el objeto se escribe en la parte superior de la tarjeta, a modo de título, en una columna a la izquierda se escriben las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad. (Castillo Oswaldo)

A continuación se muestran las tarjetas CRC correspondientes a la historia de usuario Gestionar Libro:

Tarjeta CRC	
Clase: LibroManager	
Responsabilidades	**Colaboraciones**
Obtener instancia (getInstance)	-
Registrar libros	LibroObjeto
Modificar libro	LibroObjeto
Dar baja a un libro	LibroObjeto
Obtener libros por inventario	LibroTableAccess

Tabla 6: Tarjeta CRC de la clase LibroManager.

Tarjeta CRC
Clase: LibroTableAccess

Responsabilidades	Colaboraciones
Obtener libros por inventario	-
Obtener libros por criterio	-

Tabla 7: Tarjeta CRC LibroTableAccess.

Tarjeta CRC	
Clase: LibroObjeto	
Responsabilidades	**Colaboraciones**
Obtener información del libro	-
Actualizar información del libro	-

Tabla 8: Tarjeta CRC LibroObjeto.

El resto de las tarjetas CRC pueden ser consultadas en el anexo No.2.

2.4 Codificación

La codificación debe hacerse teniendo en cuenta estándares ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad. Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crearán primero las pruebas para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco se conseguirá un desarrollo que cumpla todos los requisitos especificados. La optimización del código se debe dejar para el final, hay que hacer que la aplicación funcione correctamente y luego se optimiza. Para una mejor implementación del sistema se definieron 7 paquetes de clases fundamentales con sus respectivos componentes. El componente dataBaseLM permite que se ejecuten la autenticación en el paquete de clases Seguridad, para que una vez accedido al sistema, el componente BibliotecaUCI que permite inicializar la aplicación pasándole todos los parámetros de configuración, le comunica al componente Mediator que utilizando los componentes MenuBuilder y FxmlLocator que debe cargar el formulario MarcoPrincipal. Mediante las clases objetos, se accede a la información que brindan los componentes WrapperEntity2Object y TableAccessParent ubicados en el paquete de AccesoDatos. Esta información es mostrada por las clases que se encuentran en el paquete ClasesGUI, utilizado para manejar las interfaces gráficas. La siguiente ilustración muestra la relación descrita entre los componentes del sistema.

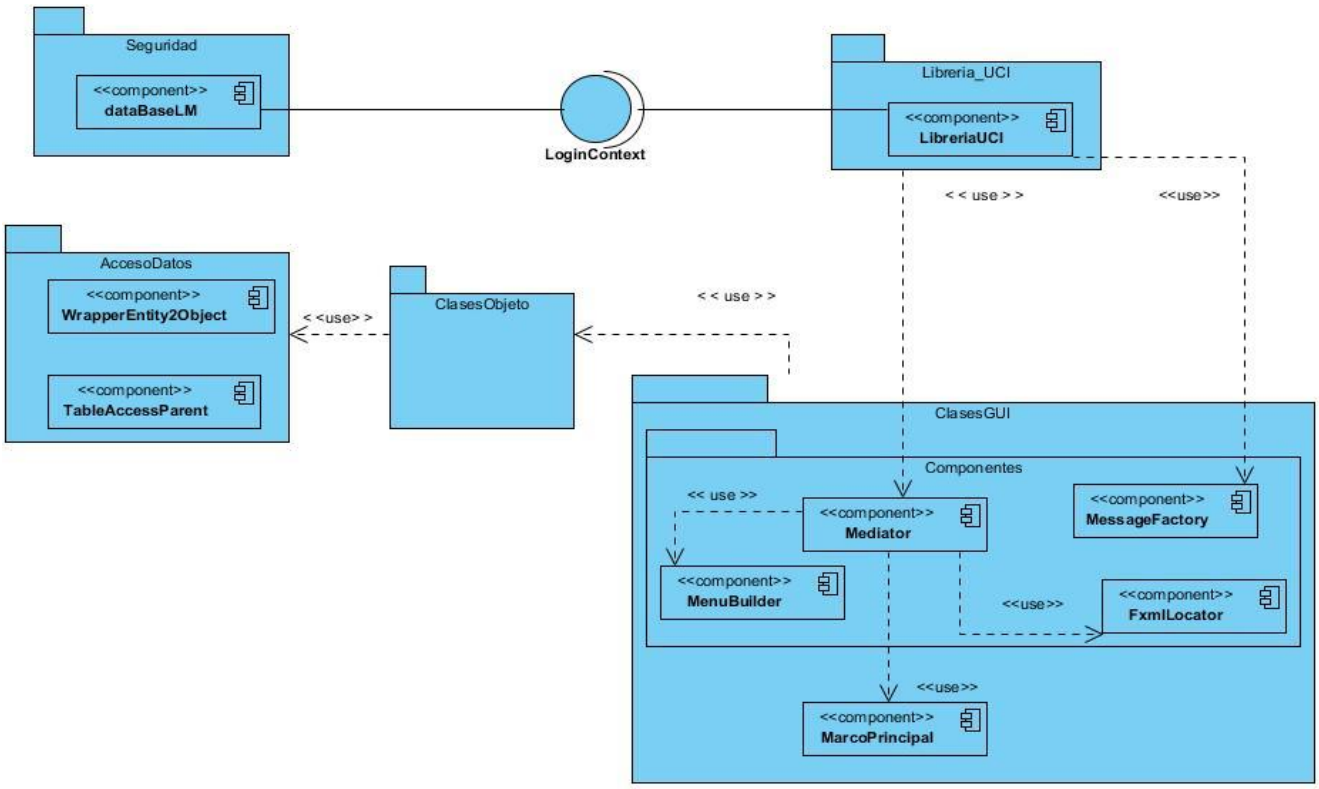


Ilustración 16: Diagrama de componentes del diseño.

2.4.1 Tareas de ingeniería por historia de usuario.

Por cada historia de usuario se definen una series de tareas que se llevarán a cabo para elaborar la funcionalidad que describe dicha historia. Las tareas pueden ser de desarrollo, de corrección o de mejora. En la descripción de cada tarea se especifica la fecha de inicio y fin de la misma, el programador responsable de cumplirla y una breve descripción de lo que hace la tarea. A continuación se detallan las tareas para la historia de usuario Gestionar Libro. El resto de las tareas se encuentran en el anexo 3.

Tarea de ingeniería	
Número de tarea: 9	**Nombre Historias de usuario:** Gestionar Libro
Nombre de la tarea: Crear interfaz para la gestión de los libros.	
Tipo de tarea: Desarrollo **(Desarrollo/Corrección/Mejora)**	**Puntos Estimados(días):** 1
Fecha inicio: 11/03/2015	**Fecha fin:** 12/03/2015
Programador responsable: Dayana Donatien Morejón	
Descripción: se rediseña la interfaz que permitirá el acceso a las acciones que permiten la gestión de los libros, como adicionar o buscar un libro.	

Tabla 9: Descripción de la tarea de ingeniería No. 9.

Tarea de ingeniería	
Número de tarea: 10	**Nombre Historias de usuario:** Gestionar Libro
Nombre de la tarea: Permitir que el usuario pueda adicionar, modificar o dar baja un libro.	
Tipo de tarea: Desarrollo **(Desarrollo/Corrección/Mejora)**	**Puntos Estimados(días):** 7
Fecha inicio: 13/03/2015	**Fecha fin:** 17/03/2015
Programador responsable: Dayana Donatien Morejón	
Descripción: en esta tarea se implementa el código que permitirá que el usuario pueda realizar las acciones de adicionar, modificar o buscar un libro.	

Tabla 10: Descripción de la tarea de ingeniería No. 10.

2.4.2 Estándares de codificación

Un estándar de codificación es el término que describe las reglas a seguir para escribir el código fuente de una aplicación en ciertos lenguajes de programación.

En el sistema informático que se propone como solución los idiomas utilizados para el nombramiento de clases, funciones y atributos fueron inglés y español, esto debido a algunas especificidades del propio lenguaje Java, así como de las especificaciones utilizadas: JPA, JAAS. Se utilizó como estilo de codificación las variantes que ofrece CamelCase. Como consecuencia el nombramiento de estos elementos de la codificación se definieron según las reglas descritas a continuación:

Nombre de las clases: en UpperCamelCase, las clases están clasificadas según su función por tanto; con el objetivo de identificarlas se hizo uso de sufijos y prefijos. Las clases controladoras se identifican utilizando el sufijo “Manager” ejemplo: LibroManager (Clase para gestionar los libros). Así mismo a las clases objeto se les puso el sufijo “Objeto” ejemplo: LibroObjeto (Clase para obtener los datos de la entidad Libro) sin embargo; en las clases entidad, por cuestiones del uso de JPA, no se utilizó la estrategia previamente mencionada.

Nombre de las funciones: en lowerCamelCase, mostrando explícitamente la operación que realiza.

Nombre de los atributos: en lowerCamelCase, su nombre muestra su rol en la clase, logrando así un mejor entendimiento de la codificación del sistema.

Conclusiones del capítulo

- Con el objetivo de satisfacer las necesidades del cliente se mantuvieron implementadas las funcionalidades encargadas de la gestión y control de literatura y la gestión de reportes, la forma de tratar los errores y la gestión del nivel de acceso de los usuarios al sistema; con la incorporación de la auditoría de los movimientos que se realicen en un periodo de tiempo determinado dentro del sistema. Consecuentemente se obtuvo un total de 11 requisitos funcionales descritos a partir de las Historias de Usuario, cuya implementación, siguiendo los estándares de codificación definidos, se estructura con el patrón arquitectónico MVC y diseña según el modelo de datos y diagrama de clases persistentes modelados.
- La utilización de patrones de diseño tales como: Singleton, Factory, Adapter, Mediator entre otros, permitieron dar soporte a un nivel de mantenibilidad aceptable para el producto final, y obtener una propuesta de solución que responda a las necesidades de desarrollo de la presente versión del sistema.

CAPÍTULO 3: PRUEBA Y VALIDACIÓN DE LA SOLUCIÓN

Introducción

En este capítulo se valora la calidad del producto de software obtenido con respecto a las tareas de la planificación y el diseño, a partir de la aplicación de métricas de diseño. Luego se realiza la verificación y validación del sistema mediante la definición de pruebas, en búsqueda del cumplimiento del objetivo que persigue la investigación.

3.1 Validación del diseño

Para validar el diseño del sistema se utilizaron métricas que se concentran en las características internas de los componentes del software e incluyen medidas de tamaño de las clases, acoplamiento y cohesión entre estas, su reutilización y complejidad; las cuales pueden ayudar a juzgar la calidad del diseño a nivel de los componentes.

Tamaño Operacional de Clase (TOC):

Utilizar esta métrica permitió medir el tamaño general de una clase, sumando el valor de la cantidad de operaciones y el número de atributos que están encapsulados dentro de esta, para así evaluar los siguientes parámetros de calidad: (Roger S. Pressman, 1997)

- ✓ Responsabilidad
- ✓ Complejidad de implementación
- ✓ Reutilización

Si el resultado obtenido al aplicar dicha métrica indica altos valores, significa que la clase tiene un alto nivel de responsabilidad, implicando que se reduzca la reutilización, sea más difícil la implementación y la realización de pruebas de dicha clase. Por tanto mientras menor sea el valor para el TOC se hará mucho más fácil la reutilización de dicha clase dentro del sistema.

El tamaño general de una clase se calcula como:

TGC= NA +TO

Donde:

NA: Número de atributos que posee la clase, tanto atributos heredados como atributos privados de la instancia.

TO: Total de operaciones que posee la clase, tanto operaciones heredadas como operaciones privadas de la instancia.

Se utilizó la siguiente tabla para medir el comportamiento de los atributos de calidad responsabilidad, complejidad de implementación y reutilización; partiendo de los umbrales pertenecientes a NA y TO de cada clase.

Atributos de calidad	Clasificación	Criterio
	Baja	Umbral<=Promedio

Responsabilidad	Media Alta	Promedio< Umbral < = 2* promedio Umbral > 2* promedio
Complejidad de implementación	Baja Media Alta	Umbral<=Promedio Promedio< Umbral < = 2* promedio Umbral > 2* promedio
Reutilización	Baja Media Alta	Umbral > 2* promedio Promedio< Umbral < = 2* promedio Umbral<=Promedio

Tabla 11: Clasificación de atributos de calidad para aplicar la métrica TOC.

Los procesos del negocio del sistema en desarrollo poseen flujos similares. Por tal motivo se escogió uno de los que mayor nivel de complejidad posee en representación del resto. Fue seleccionado entonces el flujo de clases correspondiente a la gestión de los libros en la librería.

A continuación se define la clasificación para los atributos de calidad responsabilidad y complejidad de implementación, según el umbral para un promedio P = 37.2

Baja: umbral <= 32.2 **Media:** 32.2 < umbral <= 64.4 **Alta:** umbral > 64.4

- Clasificación para Reutilización, según el umbral para un promedio P = 32.2

Baja: umbral > 64.4 **Media:** 32.2 < umbral <= 64.4 **Alta:** umbral <= 32.2

C: Clases **U:** Umbral **RP:** Responsabilidad **C:** Complejidad **R:** Reutilización

La siguiente tabla muestra las clasificaciones resultantes para el nivel de responsabilidad, complejidad de implementación y reutilización con respecto a TGC en cada una de las clases controladoras.

C	**NA**	**TO**	**U=TGC**	**RP**	**C**	**R**
Libro	13	29	42	Media	Media	Media
LibroObjeto	13	42	55	Media	Media	Media
LibroManager	3	12	15	Baja	Baja	Alta
LibroTableAccess	5	7	12	Baja	Baja	Alta
AdicionarLibro	30	7	37	Media	Media	Media

Tabla 12: Aplicación de la métrica TOC.

Como resultado de aplicar la métrica TOC se pudo observar que el nivel de responsabilidad para las clases que intervienen en el flujo de la funcionalidad registrarLibro es media para un 60 %, implicando que el 40 % de estas sean altamente reutilizables y por consecuencia que su implementación sea

medianamente compleja. Mediante las siguientes gráficas de pastel representadas a continuación se observan los resultados obtenidos.

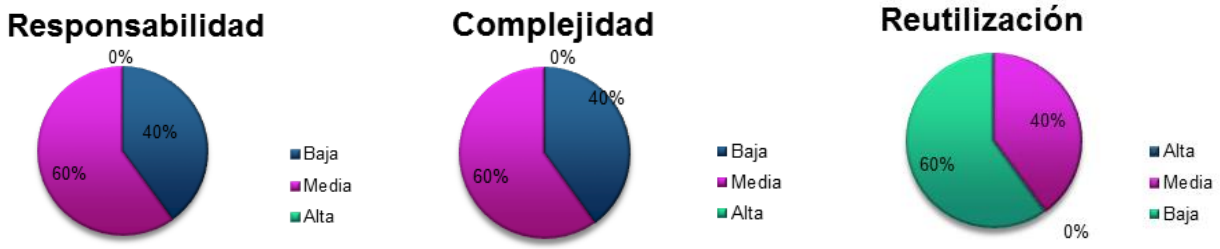


Ilustración 17: Aplicación de la métrica TOC.

Acoplamiento entre clases (AECO):

Esta métrica permite medir el acoplamiento entre clases a partir del número total de colaboraciones en listadas para una clase en su tarjeta CRC. A medida que AECO aumenta es posible que disminuya la posibilidad de reutilización. Valores elevados para esta métrica complican las modificaciones y las pruebas que aseguran que esas modificaciones se han hecho. Por tales motivos debe mantenerse el AECO en el valor más bajo posible. (Roger S. Pressman, 1997)

Para determinar el grado de afectación de los atributos de calidad que mide la métrica AECO es necesario determinar la cantidad de relaciones de uso (CRU) que posee cada una de las clases a medir, relacionadas en sus tarjetas CRC. Luego se procede a calcular el promedio de las mismas y teniendo ambos valores según los criterios expuestos en la tabla que sigue, se determina la incidencia de los atributos de calidad en cada una de las clases. (Suarez, 2012)

Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	Cantidad de relaciones de uso > 2
Complejidad del mantenimiento	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio
Cantidad de pruebas	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio

Tabla 13: Clasificación de atributos de calidad para aplicar la métrica AECO.

Dado que:

C=Clases **AC**= Acoplamiento **CM**=Complejidad de Mantenimiento **R** = Reutilización

CP=Cantidad de Pruebas

Se midió el AECO para el flujo de clases correspondiente a la gestión de los libros en la librería por ser uno de los más complejos para el negocio. Para un promedio de 1.4 CRC por cada clase se obtuvieron los siguientes resultados:

C	CRU	AC	CM	R	CP
Libro	0	Ninguno	Baja	Alta	Baja
LibroObjeto	1	Bajo	Baja	Alta	Baja
LibroManager	4	Alto	Media	Media	Media
LibroTableAccess	0	Ninguno	Baja	Alta	Baja
AdicionarLibro	3	Alto	Media	Media	Media

Tabla 14: Aplicación de la métrica AECO.

Los resultados obtenidos son aceptables debido a que el 60% de las clases pertenecientes al flujo del proceso Gestión de libros, poseen bajo o ningún acoplamiento entre sí. Esto indica que la futura complejidad de mantenimiento sea baja, se necesite realizar pocas pruebas y se facilite la reutilización de las clases.

3.2 Verificación del sistema

La verificación del sistema busca comprobar que el mismo implementa correctamente los requerimientos funcionales y no funcionales especificados. Con este fin se realizaron pruebas unitarias mediante las técnicas de Caja Blanca y Caja Negra.

3.2.1 Prueba de Caja Blanca

En la prueba de caja blanca se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos. Aunque consecuentemente esto llevaría un estudio exhaustivo, para este sistema solo se realizó una valoración de los caminos lógicos importantes (Roger S. Pressman, 1997).

Para llevar a cabo este tipo de prueba se empleó el método del camino básico. Este permitió obtener una medida de la complejidad del diseño procedimental del sistema, para ser utilizada como guía en la definición de una serie de caminos básicos de ejecución mediante el diseño de casos de prueba que garantizan que cada camino se ejecuta al menos una vez. Dado el siguiente fragmento de código enumerado perteneciente a la funcionalidad **darBajaLibro** (LibroObjeto libro, ENUM_LUGAR el) de la clase **LibroManager.**

```
public void darBajaLibro(LibroObjeto libro, ENUM_LUGAR el){
    if(el.equals(ENUM_LUGAR.SALON)){                    // 1
        libro.setCantidadSalon(0);                      // 2
    }else if(el.equals(ENUM_LUGAR.ALMACEN)){            // 3
        libro.setCantidadAlmacen(0);                    // 4
    }else if(el.equals(ENUM_LUGAR.ADMON)){              // 5
        libro.setCantidadSalon(0);                      // 6
        libro.setCantidadAlmacen(0);                    // 6
    }                                                   // 7
}
```

Ilustración 18: Código de la funcionalidad darBajaLibro (LibroObjeto libro, ENUM_LUGAR el) de la clase LibroManager.

Se obtuvo el grafo de flujo que representa el control lógico:

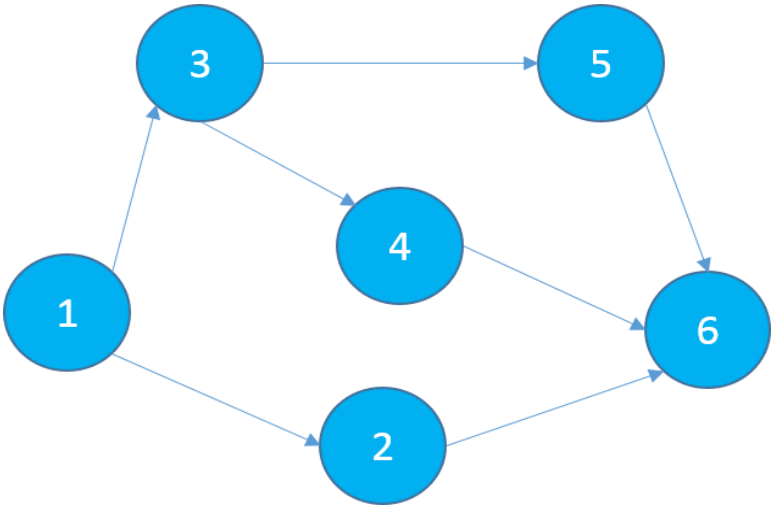


Ilustración 19: Grafo de flujo de ejecución del código de la funcionalidad darBajaLibro.

Se obtuvo la medida para la complejidad del diseño procedimental de este fragmento a partir del cálculo de la complejidad ciclomática que posee.

El resultado se pudo obtener de tres formas distintas coincidentes obteniéndose el mismo valor:

- La complejidad ciclomática coincide con el número de regiones R del grafo de flujo.
- La complejidad ciclomática, V(G), de un grafo de flujo G, se define como:
- V(G) = Aristas (A) – Nodos (N) + 2

- La complejidad ciclomática, V(G), de un grafo de flujo G, también se define como
- V(G) = Nodos de predicado (P) + 1

Donde para el caso en estudio:

- R = 3
- A – N + 2 = 7 – 6 + 2 = 3
- P + 1 = 2 + 1 = 3

Este valor obtenido indica la cantidad de caminos independientes que definió el límite de pruebas a diseñar.

Caminos:

- 1, 2, 6
- 1, 3, 4, 6
- 1, 3, 5, 6

A continuación se evidencian las pruebas realizadas:

```
    public void darBajaLibro(LibroObjeto libro, ENUM_LUGAR el){
91      if(el.equals(ENUM_LUGAR.SALON)){                          // 1
          libro.setCantidadSalon(0);                              // 2
93      }else if(el.equals(ENUM_LUGAR.ALMACEN)){                  // 3
            libro.setCantidadAlmacen(0);                          // 4
95      }else {
            libro.setCantidadSalon(0);                            // 5
97          libro.setCantidadAlmacen(0);                          // 5
98      }                                                         // 6
99  }
100
101 public List<LibroObjeto> obtenerLibroPorInventarioInicial(ENUM_LUGAR el){
102
103     return LibroTableAccess.getInstance().obtenerTotalLibrosPorInventario(el);
```

Ilustración 20: Caso de prueba para el camino 1, 2,6.

```
89
    public void darBajaLibro(LibroObjeto libro, ENUM_LUGAR el){
91      if(el.equals(ENUM_LUGAR.SALON)){                          // 1
          libro.setCantidadSalon(0);                              // 2
93      }else if(el.equals(ENUM_LUGAR.ALMACEN)){                  // 3
            libro.setCantidadAlmacen(0);                          // 4
95      }else {
            libro.setCantidadSalon(0);                            // 5
97          libro.setCantidadAlmacen(0);                          // 5
98      }                                                         // 6
99  }
100
101 public List<LibroObjeto> obtenerLibroPorInventarioInicial(ENUM_LUGAR el){
102
103     return LibroTableAccess.getInstance().obtenerTotalLibrosPorInventario(el);
```

Ilustración 21: Caso de prueba para el camino 1, 3, 4,6.

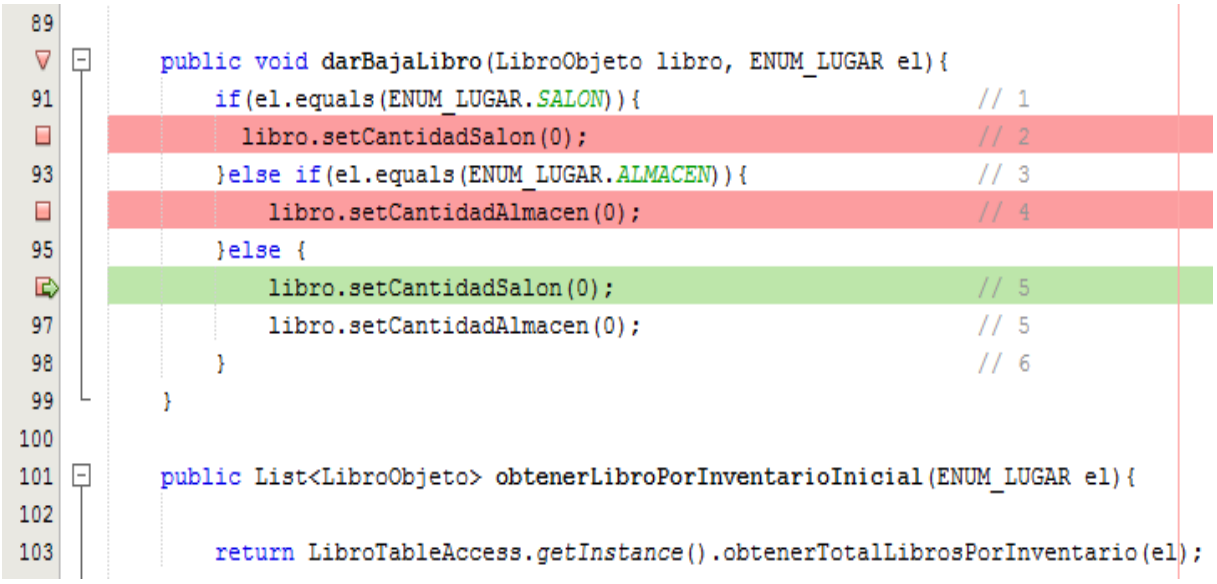


Ilustración 22: Caso de prueba para el camino 1, 3, 5,6.

Los resultados visualizan para cada caso de prueba que los caminos definidos por la complejidad ciclomática se ejecutan al menos una vez, siendo la franja verde la que indica el paso antes de completar el recorrido por el camino representado por el caso de prueba.

3.2.2 Pruebas de Caja Negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software obviando su comportamiento interno y estructura. Con estas se pretende demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma correcta.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene. (Roger S. Pressman, 1997)

Para realizar estas pruebas al Sistema para el Control de Libros se escogió el tipo de prueba de partición equivalente. Este método divide el dominio de entrada del programa en clases de datos, a partir de las cuales se derivan los casos de prueba.

Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada.

Las pruebas fueron desarrolladas por el grupo de calidad perteneciente al Centro de Gobierno Electrónico (CEGEL) con la participación de 2 probadores. Se diseñaron 10 casos de pruebas basados en requisitos, que describen los flujos pertenecientes a la gestión de usuarios y libros por ser los más críticos en el rendimiento exigido por el sistema. Se realizaron 2 iteraciones. En la primera se detectaron 21 no conformidades clasificadas de la siguiente forma:

- 11 de ortografía.
- 1 de redacción.

- 4 de funcionalidad.
- 4 de validación.

Se realizó la regresión de la primera iteración con el objetivo de comprobar la solución de las no conformidades detectadas.

En la segunda iteración se detectaron 4 no conformidades de ortografía, cuya corrección permitió en una segunda regresión liberar el sistema. El acta de liberación se encuentra en el anexo No.6.

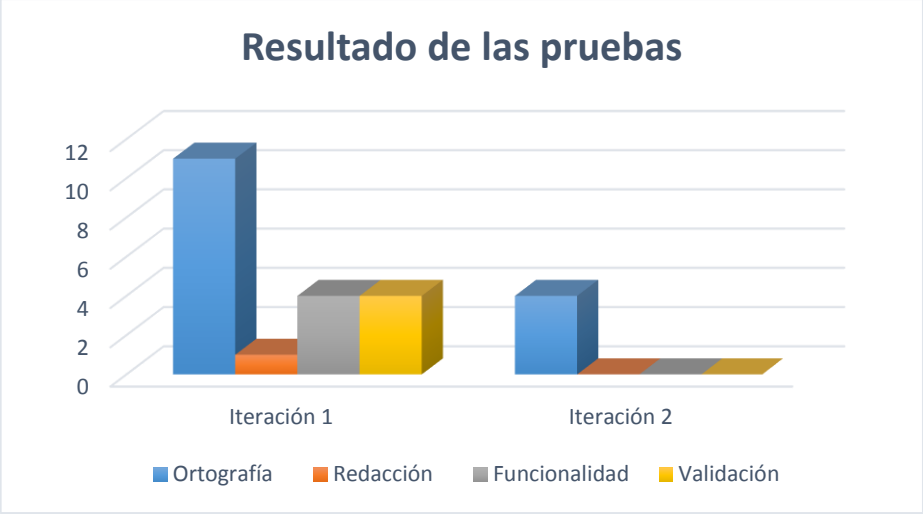


Ilustración 23: Resultados para casos de prueba de caja negra en 2 iteraciones.

A partir de los resultados anteriores queda demostrado que la herramienta implementada realiza la gestión de los libros satisfactoriamente.

3.3 Validación de las variables

La solución desarrollada está orientada a mejorar las características de mantenibilidad y usabilidad, a partir de la sustitución de tecnologías de desarrollo, implementación de nuevas funcionalidades y mejora de otras ya existentes, en el Sistema para la Gestión de Información y Control de libros para la librería de la Universidad de las Ciencias Informáticas en su versión 1.0.

Con el propósito de comprobar el cumplimiento de este objetivo, se procede a validar el comportamiento de las variables del problema de la investigación definiéndose estas de la siguiente forma:

Variable independiente:

- Las limitaciones referidas a la implementación de los requisitos funcionales y no funcionales del Sistema para la Gestión de Información y Control de libros en su versión 1.0

Variables dependientes:

- Usabilidad
- Mantenibilidad

3.3.1 Resultados de la validación de la variable usabilidad

La evaluación de la usabilidad se puede realizar a través de varios métodos o técnicas y sobre diferentes representaciones del sistema (prototipos en papel, prototipos software, sitio web implementado y otros). Existe una gran diversidad de métodos para evaluación de usabilidad, aunque únicamente se describirá aquel que se considera aplicable en el contexto real de la aplicación.

La **Evaluación Heurística** es un tipo de método de inspección que tiene como ventaja la facilidad y rapidez con la que se puede llevar a cabo. Este tipo de evaluación normalmente la desarrolla un grupo reducido de evaluadores que, en base a su propia experiencia, fundamentándose en reconocidos principios heurísticos de usabilidad, y apoyándose en guías elaboradas para tal fin, evalúan de forma independiente la aplicación, contrastando finalmente los resultados con el resto de evaluadores (Nielsen, 1990).

En la actual investigación se aplicará el modelo de evaluación heurística donde se utilizará la lista de chequeo de usabilidad propuesta en la tesis de Maestría "Procedimiento para el aseguramiento y evaluación de la usabilidad basado en patrones arquitectónicamente sensibles para los sistemas de gestión del Centro de Informatización de la Gestión de Entidades". (Leyva, 2012).

En aras de establecer una comparación, se aplicó la lista de chequeo a ambas versiones del Sistema para la Gestión y Control de Libros, para validar el cumplimiento del objetivo general de esta investigación. En la Ilustración 24 y la Tabla 15 se expone el resumen de los resultados obtenidos:

- **Evaluación de usabilidad para el Sistema para la Gestión de Información y Control de libros v1.0**

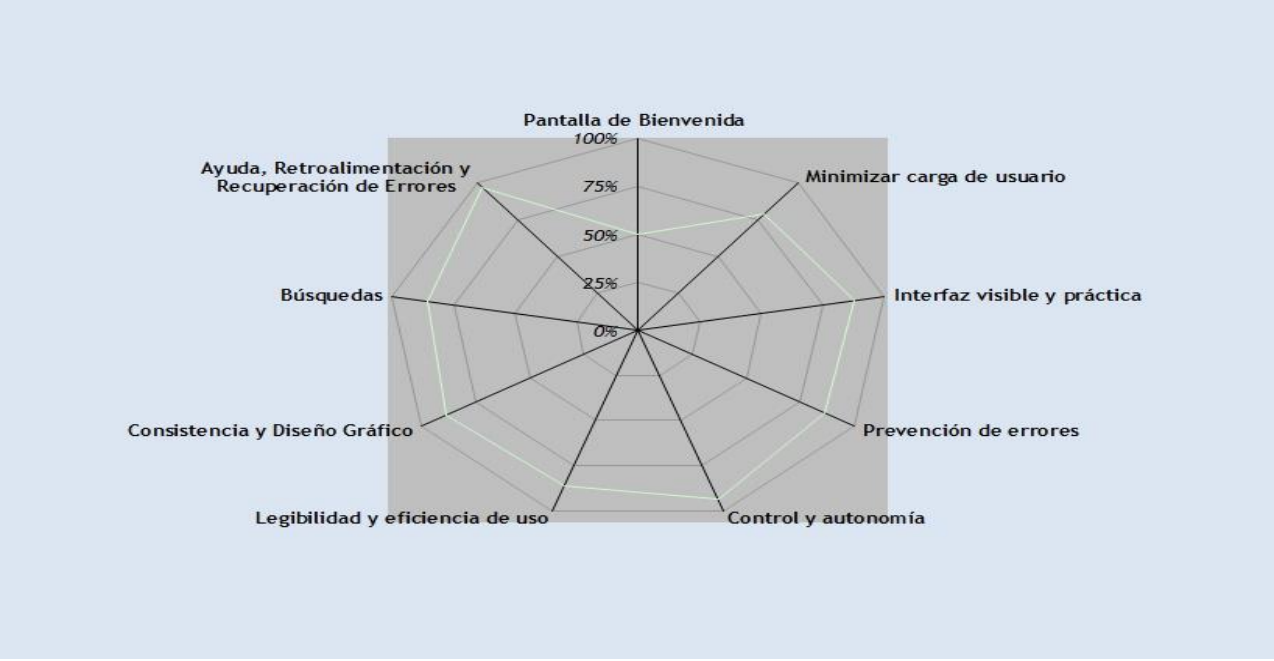


Ilustración 24: Gráfico de evaluación de la usabilidad para el SCGL v 1.0

Principios de Usabilidad	Calificación Neta	# Preguntas	# Respuesta	Calificación
Pantalla de Bienvenida	0	11	11	50%
Minimizar carga de usuario	15	27	26	79%
Interfaz visible y práctica	22	33	29	88%
Prevención de errores	13	20	18	86%
Control y autonomía	7	9	8	94%
Legibilidad y eficiencia de uso	13	21	18	86%
Consistencia y Diseño Gráfico	14	20	18	89%
Búsquedas	5	15	7	86%
Ayuda, Retroalimentación y Recuperación de Errores	27	30	29	97%
Calificación Final		186	164	84%

Tabla 15: Resumen de resultados de evaluación de usabilidad para el SGCL v 1.0.

La calificación total de usabilidad que obtuvo el sistema luego de la evaluación fue de 84%. Los aspectos que menos favorecen la usabilidad son:

- Pantalla de Bienvenida: Dado que el sistema no posee.
- Minimizar la carga del usuario: Evidenciado en el uso excesivo de ventanas para presentar la información solicitada y en la longitud de ejecución (más de 5 clics) de determinada tarea.

Otros de los aspectos que tuvo repercusión negativa fue la prevención de errores pues no se indica en los campos de texto que lo requieran, el tipo y formato de texto conveniente; entre otros elementos desfavorables.

- **Evaluación de usabilidad para el Sistema para la Gestión de Información y Control de libros v 2.0**

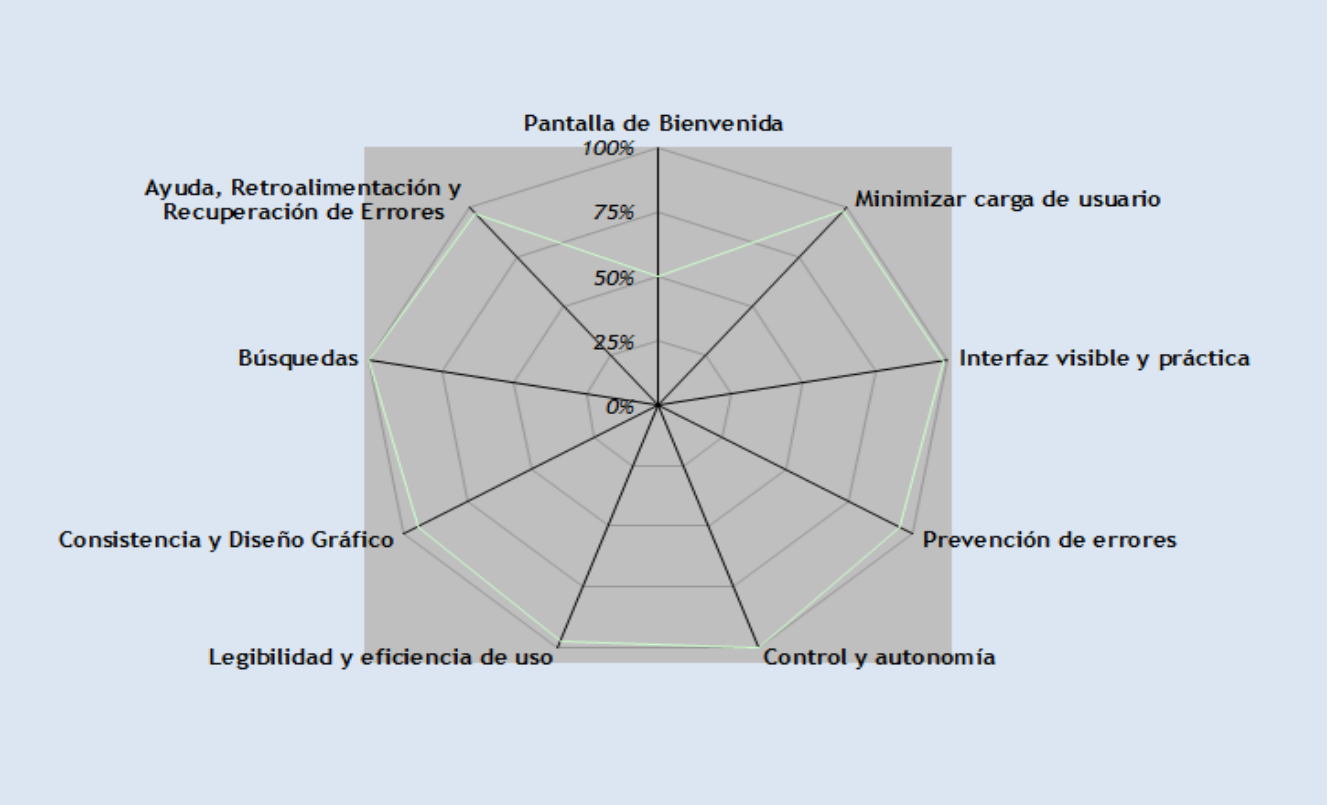


Ilustración 25: Evaluación de usabilidad para el SGCL v 2.0

Principios de Usabilidad	Calificación Neta	# Preguntas	# Respuesta	Calificación
Pantalla de Bienvenida	**0**	**11**	**5**	**50%**
Minimizar carga de usuario	**26**	**27**	**27**	**98%**
Interfaz visible y práctica	**30**	**33**	**31**	**98%**
Prevención de errores	**18**	**20**	**20**	**95%**
Control y autonomía	**9**	**9**	**9**	**100%**
Legibilidad y eficiencia de uso	**18**	**21**	**19**	**97%**
Consistencia y Diseño Gráfico	**16**	**20**	**18**	**94%**
Búsquedas	**6**	**15**	**6**	**100%**
Ayuda, Retroalimentación y Recuperación de Errores	**27**	**30**	**29**	**97%**
Calificación Final		**186**	**164**	**92%**

Tabla 16: Evaluación de la usabilidad para el SGCL v2.0

En este caso la calificación obtenida es de un 92 % de usabilidad; siendo la ausencia de una pantalla de bienvenida el principal aspecto que afecta el total cumplimiento de este criterio de calidad, entre otros de menor incidencia.

El siguiente gráfico demuestra cómo para la versión actual del Sistema para la Gestión de Información y Control de libros, la usabilidad asciende a un 92 %.

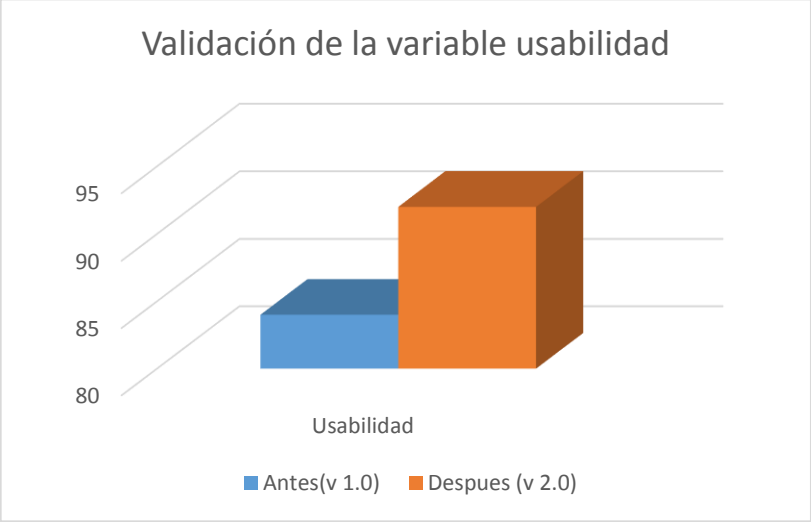


Ilustración 26: Validación de la variable usabilidad.

3.3.2 Resultados de la validación de la variable mantenibilidad

Para medir el comportamiento de la mantenibilidad de ambas versiones del Sistema para la Gestión de Información y Control de libros, se utilizó la métrica AECO.

Debido a la similitud entre los flujos de los procesos del negocio de la librería, fue seleccionado el perteneciente a la Gestión de la literatura en representación del resto, además de constituir uno de los más complejos implementados en el sistema.

- **AECO del Sistema para la Gestión de Información y Control de libros v 1.0.**

Siguiendo el procedimiento para el cálculo de los atributos de calidad: acoplamiento entre clases, complejidad de mantenimiento, reutilización y cantidad de pruebas (ver Tabla 11, epígrafe 3.1); se obtienen los siguientes resultados para un promedio de 3 CRC por cada clase:

C	CRU	AC	CM	R	CP
Libro	0	Ninguno	Baja	Alta	Baja
LibroObjeto	1	Bajo	Baja	Alta	Baja
LibroManager	6	Alto	Medio	Media	Media

LibroTableAccess	3	Alto	Medio	Media	Media
AdicionarLibro	5	Alto	Medio	Media	Media

Tabla 17: Resultado de la métrica AECO para el SGCL v 2.0

Las gráficas de pastel que a continuación se presentan, reflejan con más claridad estos resultados, los cuales permiten arribar a una conclusión sobre el comportamiento de estos atributos en el Sistema para la Gestión y Control de Libros v 1.0.

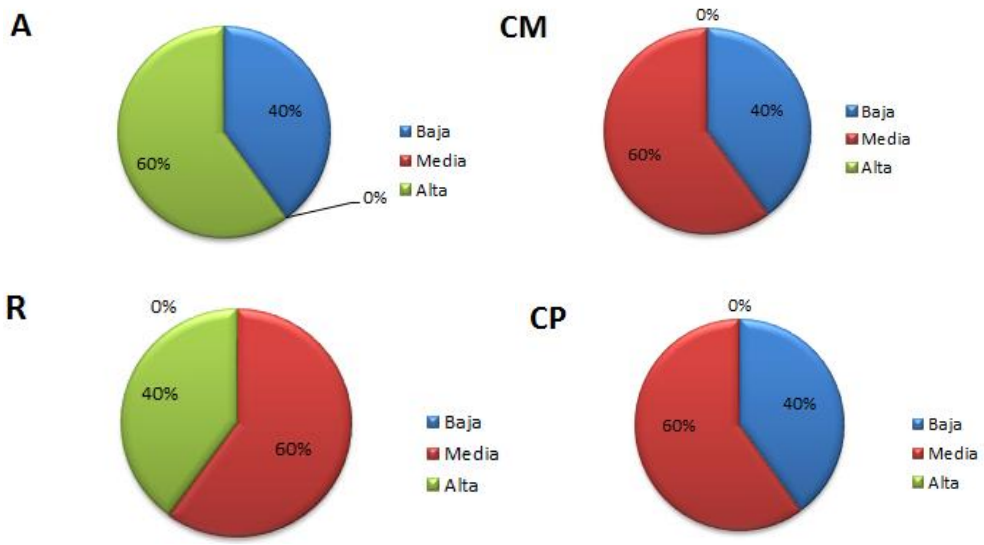


Ilustración 27: Gráficas de la métrica AECO para el SGCL v 1.0.

La siguiente tabla muestra un resumen comparativo una vez obtenidos los valores que describen el comportamiento de la mantenibilidad en ambos sistemas.

Variable	**Antes (v 1.0)**	**Después (v 2.0)**
Mantenibilidad	• Acoplamiento = 40 % Bajo • Complejidad de Mantenimiento = 40 % Medio • Reutilización = 40 % Medio • Cantidad de pruebas = 40 % Medio	• Acoplamiento = 60 % Bajo • Complejidad de Mantenimiento = 60 % Baja • Reutilización = 60 % Alta • Cantidad de pruebas = 60 % Baja

Tabla 18: Comparación de la mantenibilidad para el SGCL v 1.0 y v 2.0.

Observados estos datos, es evidente la evolución del sistema hacia la versión 2.0, con respecto al tratamiento de los parámetros de calidad evaluados. Esta conclusión se sustenta con la diferencia existente entre los valores obtenidos para cada uno de los sistemas.

Se obtuvo que para el nuevo sistema el acoplamiento dependiente de la cantidad de clases que lo condicionan, 20 % más bajo que en la versión anterior, donde por consecuencia el nivel de reutilización lo supera en un 20 %. Paralelamente, la cantidad de pruebas necesarias y la complejidad de mantenimiento, descienden a un 60 %.

Conclusiones del capítulo

- A partir de los resultados obtenidos mediante las métricas TOC y AECO para la validación del diseño del SGCL v 2.0, se evidencia que el nivel de responsabilidad para los componentes es medio, representado por el 60 % de estos, manteniéndose la complejidad de implementación al margen de este valor, e implicando que la reutilización entre los mismos sea alta para el 40%. Se obtuvo además una medida baja para el acoplamiento entre las clases del diseño, propiciando la alta reutilización entre estas y la realización de la menor cantidad de pruebas.
- Como resultado de las pruebas empleadas para verificar que el sistema se ha estado construyendo correctamente, se obtuvo mediante la prueba de Caja Blanca , una medida que permitió definir los caminos lógicos de ejecución más importantes , donde mediante casos de prueba definidos, se garantizó que estos se ejecutaran al menos una vez.
- Quedó demostrado mediante las pruebas de Caja Negra realizadas en una segunda iteración, que las funcionalidades de SGCL 2.0 son operativas, donde las entradas son aceptadas y las salidas correctas, manteniéndose la integridad de la información externa.
- La prueba de usabilidad realizada permitió arribar a la conclusión de que el SGCL v 2.0 posee un nivel de usabilidad superior que el de la versión anterior.
- Con la validación de la mantenibilidad mediante la métrica AECO, se evidencia que la implementación de la versión 2.0, cumple con las expectativas para esta variable de investigación; debido a que la solución implementada provee niveles superiores con respecto a la versión anterior.

CONCLUSIONES GENERALES

- El análisis de las características funcionales y tecnológicas del Sistema para la Gestión de la información y Control de Libros v1.0, permitió detectar aspectos que mediante la implementación de cambios, podían ser mejorados o sustituidos, en aras de alcanzar niveles de usabilidad y mantenibilidad superiores a los existentes; motivo este que conduce a la propuesta de una nueva versión del sistema.
- La selección de una metodología para el proceso de desarrollo, permitió estructurar el diseño e implementación de la solución mediante fases, obteniendo en cada una los artefactos necesarios para la descripción y modelación del negocio.
- El estudio de las herramientas, tecnologías y lenguajes para el desarrollo de software propició la selección de las idóneas para la implementación de la solución según las necesidades que caracterizan el problema de la investigación.
- La utilización de patrones de diseño contribuyó en gran medida al soporte del bajo acoplamiento entre las clases que componen el diseño, disminuyendo su complejidad de implementación y por consiguiente elevando la reutilización de las mismas.
- El empleo de métricas y pruebas de calidad del software, demostraron que el Sistema para la Gestión y Control de la información de Libros v 2.0, posee niveles de usabilidad y mantenibilidad superiores a la versión anterior.

RECOMENDACIONES

- Incrementar el número de funcionalidades del sistema, con el objetivo de satisfacer nuevas necesidades del cliente.
- Seguir en la búsqueda de mecanismos de implementación y diseño que permitan aumentar los niveles de usabilidad y mantenibilidad.
- Implementar los procesos Gestión de Ventas y Gestión de Recursos Humanos,
- Incorporar al sistema funcionalidades que permitan la entrada de información a partir de un archivo en formato CSV.

GLOSARIO DE TÉRMINOS

Look and Feel: Se utiliza en relación a una interfaz gráfica de usuario e incluye aspectos de su diseño, incluyendo elementos como los colores, formas, diseño, y tipos de letra, así como el comportamiento de los elementos dinámicos como botones, cajas y los menús.

CamelCase: Es un estilo de escritura que se aplica a frases o palabras compuestas.

UpperCamelCase: Cuando la primera letra de cada una de las palabras es mayúscula.

LowerCamelCase: La primera letra de cada una de las palabras es minúscula.

Change listeners: Objetos oyentes para eventos relacionados a las entidades persistentes.

BIBLIOGRAFÍA

AENOR. 2001.
http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0024329. http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0024329. [En línea] 28 de febrero de 2001.

Alpaev, Sergiy. 2006. UDLAP-Bibliotecas. 2006.

Carlos Santander Vega, Bernhard Hitpass Heyl. 2013. ResearchGate. [En línea] 2013. http://www.researchgate.net/profile/Bernhard_Hitpass.

Castillo Oswaldo, Figueroa Daniel, Sevilla Hector. programacionextrema.tripod.com. [En línea] [Citado el: 10 de Marzo de 2015.] http://programacionextrema.tripod.com.

Community. 2010. Community, NetBeans. 2010. NetBeans. NetBeans. . [En línea] 2010. http://netbeans.org/. .

Craig Larman. 1999. Introducción al análisis y diseño orientado a objetos. Parte IV Fase de diseño. [aut. libro] Craig Larman. UML y Patrones. México : PEARSON, 1999.

Don Wells. 2013. Extreme Programming. Extreme Programming. [En línea] 8 de Octubre de 2013. [Citado el: 25 de Febrero de 2015.] http://www.extremeprogramming.org.

Eliza Bertino, Lorenzo Martino. 1995. Sistemas de base de datos orientada a objetos. . Estados Unidos de América : Ediciones Diaz Santos S.A, 1995.

Ernst, Timo. 2011. Performance Analysis and Acceleration for Rich Internet Application Technologies∗. 2011.

Frías, Roberley Cuadra. 2012. Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio. . La habana : s.n., 2012.

Galilea Rhode. 2013. Estudio de mercado lilbrería cristiana Pepelapiz Guatemala: s.n., 2013.

Grau, Xavier Ferré. 2000. Principios Básicos de Usabilidad para Ingenieros Software. 2000.

Guevara de. Desc., Jorge Martinez Ladrón. 2012. Fundamentos de programación en Java. s.l. : ISBN 978-84-96285-36-2, 2012.

Ian Somerville. 2005. Desarrollo rápido de software. [aut. libro] Ian Somerville. Ingeniería del Software. Madrid : Pearson Educación, S.A, 2005.

Isasias Carrillo Perez, Rodrigo Perez González, Aureliano David Rdriguez Martin. 2008. Metodología del desarrollo del Software. 2008.

ISO. 2015. ISO 25000 calidad del producto de software. ISO 25000 calidad del producto de software. [En línea] 2015.

Jose H. Canos, Patricio Letelier, Mª Carmen Penades. 2003. Metodologías Ägiles de Desarrollo de Software. Valencia : s.n., 2003.

Juan Carlos Perez Gonzalez. 2012. dawdamasir.com. [En línea] 5 de 2012. http://dawdamasir.com/wp-content/uploads/2012/05/UD8.ISO_.pdf.

Krishnamurthy. 2011. umsl.edu. [En línea] 2011. http://www.umsl.edu/~sauterv/analysis/F08papers/View.html#Introduction_8121203202754259_.

Leyva, Iliannis Pupo. 2012. Procedimiento para el aseguramiento y evaluación de la usabilidad basado en. La habana : s.n., 2012.

Mikael Berndtsson, Jörgen Hanssonn, Björn Lundell y Olsson, Björn. 2008. Thesis Projects. Londres : Springer, 2008. ISBN-13: 978-1-84800-008-7.

Nielsen, Jakob. 1990. Heuristic evaluation of user interfaces. Dinamarca : s.n., 1990. ACM O-89791 - 345-O/90/0004-0249 .

OMG. 2011. OMG. [En línea] 03 de de enero de 2011. http://www.omg.org/spec/BPMN/2.0/.

Oracle. 2013. https://docs.oracle.com/javafx/2/swing/overview.htm. [En línea] 2013.

—. 2015. NetBeans. NetBeans. [En línea] 2015. https://netbeans.org/community/releases/80/.

Oracle Technology Network. 2014. http://www.oracle.com/technetwork/es/java/javase/downloads/index.html. [En línea] 2014.

P. J. Deitel, H. M. Deitel. 2006. Java, how to program . New Jersey : Pearson, 2006.

Roger S. Pressman. 1997. Ingenieria del software: un enfoque practico. 1997.

Rolando Alfredo Hernández León, Sayda Coello. 2011. El proceso de la investigacion cientifica. La Habana : Editorial Universitaria, 2011. ISBN 978-959-16-1307-3.

Suarez, Laritza Rodríguez. 2012. Sistema de control de libros para la librería de la Universidad de las Ciencias Informáticas version 1.0. 2012.

The Apache Software Foundation. 2014. Apache Derby. Apache Derby. [En línea] 2014. [Citado el: 10 de 2 de 2015.] http://db.apache.org/derby/index.html.

timo-ernest. 2010. (http://www.timo-ernst.net/misc/riabench-start/. [En línea] 15 de Septiembre de 2010.