Universidad de las Ciencias Informáticas

Facultad 2

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

Título: Aplicación informática para la migración dinámica entre formatos bibliográficos

Autor(es):

Dayna Caridad Hernández Flores

Andrés Sellés González

Tutor(es):

Ing. Heliodoro Rodríguez Milián

Ing. Rolando Gual Pérez

Consultante:

Dr. Amed Abel Leiva Mederos

Ciudad de La Habana, Junio 2013

Año 55 de la Revolución

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo titulado: “Aplicación informática para la migración dinámica entre formatos bibliográficos”, y otorgamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

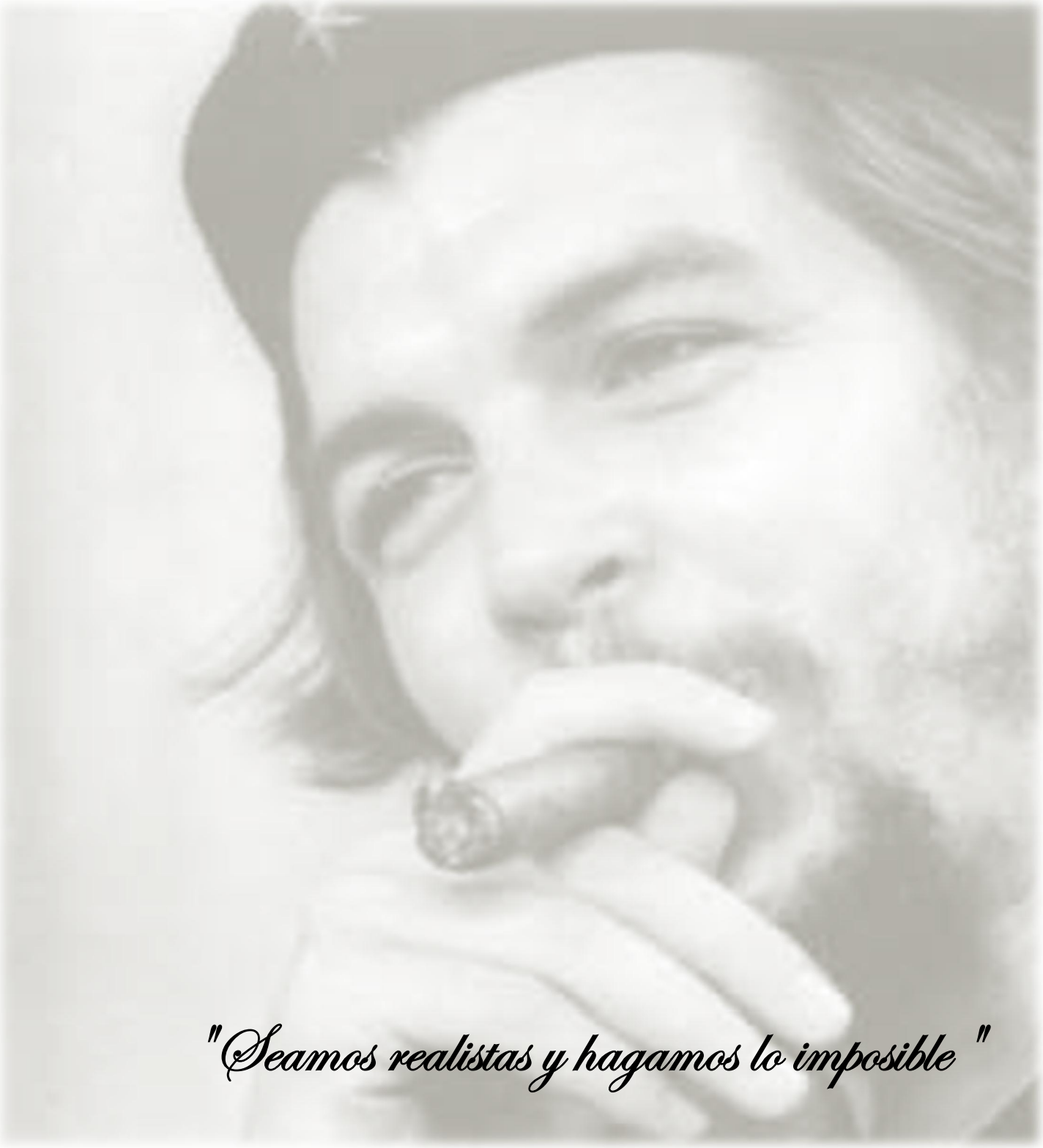
Para que así conste firmamos el presente documento a los ____ días del mes de _____ del año _____.

_____ _____

Dayna Caridad Hernández Flores Andrés Selles González

_____ _____

Ing. Heliodoro Rodríguez Milián Ing. Rolando Gual Pérez


"Seamos realistas y hagamos lo imposible "

Dedicatoria

A mi mama por darme la vida y por hacerme la mujer que soy. Por brindarme esa fuerza que me ayuda a levantarme cada día y seguir adelante. Por apoyarme en todos los momentos difíciles de mi vida y por ayudarme a hacer realidad todos mis sueños.

A la persona que me ayuda a sentirme responsable y segura de mis decisiones, mi papá. Gracias por estar a mi lado en todo momento, por acompañarme en los momentos difíciles y aconsejarme sobre las malas decisiones que he tomado en mi vida. No me alcanzan las palabras para agradecerte cada momento de mi vida.

A mi hermana, que es madre y amiga. Por apoyarme siempre y cuidar de mí, por darme su cariño y defenderme ante el mundo y por sentirse siempre orgullosa de mi. Nena, sabes que eres mi modelo a seguir como mujer, como persona y como profesional.

Dayna

Agradecimientos

A papi y mami por apoyarme en este sueño y en todos los demás.

A mi hermana por preocuparse por mí y por disfrutar y sufrir conmigo cada momento de estos 5 años y de mi vida.

Agradezco a mi novio Victor, por apoyarme desde que entre en esta universidad. Gracias a ti logre superar todos los obstáculos en estos 5 años. Te Quiero.

Agradezco a mi compañero Selles, por aguantarme y por ayudarme a lograr este sueño.

Agradezco a Rosy, Elva y Yeni, por vivir tantas locuras a mi lado y por brindarme su ayuda en los momentos difíciles.

A Gual, por guiarnos y darnos ánimo en todo momento.

Dayna

Resumen

Las bibliotecas atesoran una gran diversidad de materiales bibliográficos que son puestos a disposición del público en general. Actualmente los procesos bibliotecarios son llevados a cabo a través de los Sistemas Integrados de Gestión Bibliotecaria (SIGB) encargados de la adquisición, conservación y divulgación de la información. Pero en muchas ocasiones cuando se desean migrar los registros de un sistema a otro existen problemas de incompatibilidad entre los formatos bibliográficos que estos emplean. Por tal motivo se necesita un mecanismo que les permita la migración de sus bases de datos logrando la integridad de los registros existentes. A partir de esta situación se define la implementación de un software que permita la migración dinámica entre formatos bibliográficos aplicada al SIGB basado en KOHA, para evitar que se pierdan registro bibliográficos que tomaron años en ser catalogados.

El presente trabajo se enfocó en el diseño e implementación de una aplicación informática que se encargara de la migración dinámica entre formatos bibliográficos, nombrada Conversor. Esta fue desarrollada a partir de las funcionalidades definidas por el cliente y la arquitectura diseñada por el equipo de trabajo. Durante la investigación se realizó el análisis de la documentación para el estudio de soluciones existentes que reflejó la necesidad de implementar una solución, partiendo de un análisis para el estudio de las tecnologías, metodología, herramientas y lenguajes en los que se basa el desarrollo de la aplicación.

Palabras claves: *SIGB, formatos bibliográficos, registros bibliográficos, migración de datos*.

Índice

Índice

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

La historia de la humanidad se ha edificado sobre la base de la recopilación de los conocimientos adquiridos en disímiles soportes a través de los años. El ascenso en la producción de documentos ha traído consigo que los procesos de organización y recuperación se vayan tornando más complejos. La necesidad de salvaguardar y organizar todo el valioso patrimonio que conforma la información, fue el detonante para el surgimiento de las primeras bibliotecas del mundo.

El vocablo "biblioteca" proviene del griego ***biblion*** significa libro y ***teca*** significa caja, lo cual puede traducirse etimológicamente como el lugar donde se guardan los libros. Las primeras bibliotecas de mundo fueron construidas en los templos de importantes ciudades de la antigua Mesopotamia. Luego surgen importantes bibliotecas que marcan el período helenístico: la Biblioteca de Alejandría y la Biblioteca de Pérgamo (1).

La información se ha convertido en uno de los bienes más valuados para garantizar el crecimiento económico, cultural y social de la humanidad. Por tales motivos en un mundo donde la información es poder, resulta imprescindible contar con bibliotecas actualizadas y con recursos modernos para administrarlas.

El surgimiento de las Tecnologías de la Información y las Comunicaciones (TIC), ha permitido que las bibliotecas inicien un proceso de digitalización de sus recursos y procesos. Las TIC se han transformado en herramientas imprescindibles para el intercambio de información y la mejora de los servicios en los centros bibliotecarios. Con el fin de lograr la automatización de los procesos y los servicios de estas instituciones, surgen los Sistemas Integrados de Gestión Bibliotecaria (SIGB). Estos sistemas utilizan estándares para una adecuada organización y codificación de la información. Al ordenamiento y codificación particular de los datos se conoce como "formato bibliográfico". En tal sentido, el formato bibliográfico se define como la estructura y el sistema de códigos que hacen comprensible a la máquina el contenido de un documento bibliográfico.

La Universidad de las Ciencias Informáticas cuenta con un Sistema Integrado de Gestión Bibliotecaria basado en KOHA que se encarga de la adquisición, conservación y divulgación de un gran volumen de obras y bibliografías que responden a una amplia gama de géneros. Dicho sistema puede ser desplegado

en tres escenarios fundamentales: en entidades donde no existe un sistema de gestión bibliotecaria, en entidades donde existe un sistema de gestión bibliotecaria con un formato bibliográfico compatible con el del sistema que se desea implantar, y por último, en instituciones con experiencia en el uso de sistemas de gestión bibliotecaria, donde exista incompatibilidad entre formatos bibliográficos al desplegar un nuevo sistema. Este último escenario es característico de instituciones que han gestionado información bibliográfica durante muchos años y necesitan un sistema que les permita mejorar sus procesos y servicios. Sin embargo, esto acarrea dificultades al no poseer un mecanismo que permita la utilización en el sistema implantado de los registros existentes. Esta incompatibilidad entre formatos bibliográficos puede ocasionar la pérdida de registros que tomaron años en ser catalogados y ocasionar dificultades en los procesos de la institución.

En función de lo anteriormente planteado se propone como **problema a resolver:**

¿Cómo contribuir al aprovechamiento de los registros existentes en los sistemas bibliográficos cuando se desea implantar un sistema integrado de gestión bibliotecaria?

Con el propósito de dar solución a este problema se traza como **objetivo general** desarrollar una aplicación informática que permita la migración dinámica entre diferentes formatos bibliográficos, definiéndose para su correcto cumplimiento los siguientes **objetivos específicos:**

1. Elaborar el marco teórico orientado a las herramientas para la migración dinámica entre formatos bibliográficos.
2. Realizar el análisis y el diseño de la solución para la migración dinámica entre formatos bibliográficos.
3. Implementar la herramienta respetando la arquitectura definida para la misma.
4. Validar el correcto funcionamiento de la solución.

Para dar solución al problema descrito se planteó como **idea a defender**: “Si se desarrolla una aplicación informática que permita la migración dinámica entre formatos bibliográficos se contribuirá a la exportación e importación de registros bibliográficos, reduciendo el tiempo que es empleado para la realización de este proceso y evitando que se pierdan registros que tomaron años en ser catalogados”

Se proponen para este fin un conjunto de **tareas de investigación** para apoyar la solución de la temática planteada**:**

1. Estudio de los principales conceptos asociados a los SIGB y los formatos bibliográficos para conocer los elementos técnicos que faciliten la comprensión del tema.
2. Estudio de los formatos bibliográficos más utilizados para comprender su funcionamiento.
3. Estudio de los procesos de migración entre formatos bibliográficos para realizar una correcta conversión entre los mismos.
4. Estudio de las herramientas informáticas existentes de conversión entre formatos bibliográficos para conocer sus principales características.
5. Estudio de las tecnologías, herramientas y metodología a emplear para la implementación de la solución.
6. Estudio de los patrones de diseño y arquitectura para mitigar los problemas que puedan surgir en la codificación de la solución.
7. Descripción de la arquitectura del sistema para mostrar la estructura global del software.
8. Realización de pruebas funcionales a la solución.
9. Validación de la herramienta usando formatos bibliográficos del sistema ABCD y KOHA.

El **objeto de estudio** comprende el proceso de migración entre formatos bibliográficos y el **campo de acción** abarca el proceso de migración entre formatos bibliográficos en SIGB KOHA y ABCD.

Para llevar a cabo las tareas de investigación se emplearon los siguientes métodos de investigación:

Métodos Teóricos:

- *Analítico–Sintético*: En el presente trabajo se lleva cabo el análisis de las teorías y documentos relacionados con la migración de datos, permitiendo la extracción de los elementos más importantes que se relacionan con este tema.

- *Sistémico*: Se tiene en cuenta el problema como un todo, con el propósito de que se desarrolle un sistema flexible y robusto que cumpla con los requisitos exigidos.

Métodos Empíricos:

- *Observación:* En el contexto de este trabajo se realizará una adecuada observación de la información de cada uno de los conceptos relacionados con migración de datos, obteniendo un conocimiento más profundo sobre el tema.

La estructura del documento consta de cuatro capítulos y se abarcan los siguientes temas:

CAPÍTULO 1. Fundamentación Teórica: Este capítulo incluye una descripción de los principales conceptos a tratar: SIGB, formatos bibliográficos, migración de datos, y un análisis del estado del arte de los sistemas similares. Se describirán los lenguajes de modelado, metodologías de desarrollo de software y herramientas utilizadas en la solución del problema.

CAPÍTULO 2. Características del Sistema: En este capítulo se caracteriza el sistema: se plantean a grandes rasgos las Historias de Usuarios (HU) que son de interés para la primera entrega del producto así como una propuesta del prototipo no funcional de la aplicación.

CAPÍTULO 3. Diseño, Desarrollo y Prueba del Sistema: Según la metodología de desarrollo utilizada, este capítulo se refiere a las propias fases de la misma, definiendo la arquitectura del sistema y con ello patrones arquitectónicos empleados. Se procederá a abordar las HU definidas y tareas de la ingeniería en cada iteración para luego hacer una revisión de las mismas.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

El presente capítulo abarca los conceptos fundamentales referentes a la migración entre formatos bibliográficos en los SIGB. En el mismo se mencionan algunos SIGB utilizados a nivel mundial, así como los formatos bibliográficos más utilizados por estos sistemas. Además se realiza un estudio del proceso de migración de datos y las metodologías utilizadas para ello. Por otra parte, se realiza la selección de los lenguajes, tecnologías y metodología a emplear para desarrollar el software de migración dinámica entre los distintos formatos bibliográficos escogido para la solución.

1.1. Marco Conceptual

1.1.1. Sistemas Integrados de Gestión Bibliotecaria

Un SIGB es *"un conjunto organizado de recursos humanos que utilizan dispositivos y programas informáticos, adecuados a la naturaleza de los datos que deben procesar, para realizar procesos y facilitar los servicios que permiten alcanzar los objetivos de una institución bibliotecaria: almacenar de forma organizada el conocimiento humano contenido en todo tipo de materiales bibliográficos para satisfacer las necesidades informativas, recreativas y de investigación de los usuarios*" (2). Es un software diseñado para automatizar una o más funciones propias de una biblioteca. Además comparte una base de datos común, la cual se interrelaciona con los programas que realizan las diferentes funciones del proceso documental.

- **ABCD**

  El Sistema de Automatización de Bibliotecas y Centros de Documentación (ABCD) se desarrollan como iniciativa de BIREME[1] con el apoyo del consorcio de Universidades de Flandes en Bélgica y está dirigido principalmente a países en vías de desarrollo. Es un sistema multilingüe que pertenece a la

[1] **BIREME**: Biblioteca Regional de Medicina Centro Latinoamericano y del Caribe de Información en Ciencias de la Salud.

familia de software ISIS y se distribuye con bases de datos MARC 21 y CEPAL. No obstante, permite el uso de cualquier formato de catalogación creando una nueva base de datos en ISIS.

El sistema ABCD integra los siguientes módulos:

- Adquisición
- Circulación/Préstamo
- Estadísticas
- Administración de BD y usuarios
- Publicaciones periódicas.
- Catalogación.

➢ **KOHA**

KOHA[2] es el primer software libre utilizado en la automatización de bibliotecas, creado en 1999 por Katipo Communications para la Horowhenua Library Trust en Nueva Zelanda. Este SIGB permite administrar los procesos bibliotecarios y gestionar los servicios a los usuarios (3). Se integra por 6 módulos: administración, catalogación, adquisiciones seriadas, circulación, autoridades y catálogos al público. La aplicación gestiona también adquisiciones, el control de autoridades y permite la captura de registros bibliográficos transmitidos desde clientes Z39.50[3]. Además es un programa multilingüe con traducciones disponibles en inglés, francés, español, polaco y chino.

➢ **NewGenLib**

NewGenLib es un sistema integrado de gestión bibliotecaria desarrollado por Vero solutions Pvt Ltd[4] apoyado por el Instituto Kesavan de información y gestión del conocimiento en Hyderabad en India. El primer lanzamineto de este sistema fue en marzo de 2005, aunque no fue hasta el 2008 que fue declarado software libre bajo la licencia GPL. Este es un software basado en la web y posee una

[2] Proviene del idioma maorí que significa obsequio o donación

[3] Marta Arango menciona que el Z39.50 "es un estándar que especifica un conjunto de reglas para gestionar las formas y procedimientos de interconexión remota de computadoras, con el propósito de buscar y recuperar información sin importar el servidor en el que se encuentre" (39).

[4] Private Limited Company o Compañía Limitada Privada.

arquitectura de múltiples niveles mediante Java, un servidor de aplicación basado en J2EE[5] y la base de datos del servidor es PostgresSQL. Es comptaible con el formato MARC 21, cuenta con un editor de MARC y permite la importación de datos bibliográficos y de autoridad en plantillas de catalogación. NewGenLib es un sistema multiplataforma que actualmente se encuentra distribuida su versión mas reciente NewGenLib2.4 (4).

- **OpenBiblio**

OpenBiblio es un sistema integrado de gestión bibliotecaria de código abierto que contiene un catálogo en línea, una sección para préstamos, catalogación y una funcionalidad que permite la administración de personal. Fue creado por Dave Stevens en al año 2002 utilizando el lenguaje de programación PHP. Permite realizar alertas al usuario a traves de mensajes prediseñados, el control de multas por devolución tradía de los libros y brinda modelos estadísticos para el uso de la biblioteca y el material. El sistema ha sido traducido al español y es utilizado en el sistema de educación primaria en Chile, aunque otros países como Cuba y Venezuela han expresado su interés en este sistema (5).

1.1.2. Formatos bibliográficos

Los formatos bibliográficos son normas que indican que datos deben ser empleados para describir un material bibliográfico y en qué forma se ingresan dichos datos. Su propósito es facilitar el entendimiento con los sistemas de información y el manejo y recuperación de los datos. Existen muchas clases y niveles de formatos bibliográficos. Su variedad depende no solo del tipo de documentación que intentan reflejar, sino también de los resultados que se pretende conseguir al aplicarlos (6).

1.1.3. Desarrollo de los formatos bibliográficos

La creación de un formato bibliográfico es producto de las condiciones específicas en que se realiza la catalogación en las bibliotecas y de la necesidad de orientar esta práctica en correspondencia con los

[5] Java 2 Platform, Enterprise Edition

estándares internacionales y las posibilidades de intercambio de registros bibliográficos entre las instituciones de todo el mundo a partir de la infraestructura proporcionada por Internet.

La red de bibliotecas adoptó -en sus inicios- el formato bibliográfico CEPAL del sistema de información de la Comisión Económica para América Latina y el Caribe (CEPAL) con el propósito de que sirviera como estructura para cualquier base de datos bibliográfica que se desarrollara dentro del sistema, incluido el catálogo automatizado de las bibliotecas. Aunque dirigido a la unificación del diseño de las bases de datos bibliográficas, dicho formato presenta el inconveniente de que no resulta apropiado para la organización de los datos bibliográficos, según los estándares internacionales.

En la actualidad, la comunidad bibliotecaria ha basado la automatización de sus catálogos en la familia de formatos MARC (IBERMARC, UNIMARC, MARC 21, MARCXML). Estos formatos reflejan, en su estructura, la organización que sustentan las Reglas de Catalogación Angloamericanas (2da Edición), código adoptado por la mayoría de las bibliotecas para la catalogación.

1.1.4. Automatización de los formatos bibliográficos

- **CEPAL**

La Comisión Económica para América Latina y el Caribe (CEPAL) ha desarrollado desde 1981 el Sistema de Información Bibliográfica para generar y mantener bases de datos bibliográficas relacionadas con la automatización de las bibliotecas. De esta forma surge el formato CEPAL, propuesto por la comisión con el fin de crear un formato normalizado para procesamiento de información bibliográfica (7).

- **MARC**

En la década de 1960 surge MARC[6] como un estándar digital internacional de descripción de información bibliográfica. Fue desarrollado con el fin de facilitar la creación y diseminación de catalogación computarizada de una biblioteca a otra dentro de un mismo país o entre varios países. Los estándares MARC definen tres aspectos de un registro MARC: la estructura de registros, las designaciones de campos dentro de cada registro y el propio contenido del registro.

[6] MAchine-Readable Cataloging o Registro Catalográfico Legible por Máquina

En el mundo existen varias versiones de MARC, entre ellas: MARC 21, UNIMARC y MARC-XML (8).

- **UNIMARC**

La IFLA en su intento normalizador internacional en el ámbito bibliográfico, publicó en 1977 un formato internacional para el intercambio de datos bibliográficos autorizados, que se denominó formato Universal MARC (UNIMARC). Con él se pretende agilizar y mejorar la transferencia de información evitando los problemas que suponen los distintos formatos nacionales de intercambio.

Los campos de datos se agrupan en bloques de acuerdo con el primer caracter de la etiqueta, la función de los datos dentro del registro está determinada generalmente por el bloque al que pertenece, como se muestra a continuación (9):

0XX Bloque de Identificación.
1XX Bloque de información codificada.
2XX Bloque de información descriptiva.
3XX Bloque de notas.
4XX Bloque de enlace de entrada
5XX Bloque de títulos relacionados.
6XX Bloque de análisis de asunto.
7XX Bloque de responsabilidad intelectual.
8XX Bloque de uso internacional.

UNIMARC establece los identificadores de contenido de los registros bibliográficos (etiquetas, indicadores y códigos de subcampos) así como el formato lógico y físico. Es aplicable a monografías, publicaciones periódicas, material cartográfico, música impresa, grabaciones sonoras y vídeo.

- **MARC 21**

MARC 21 surge como resultado de la combinación entre los formatos MARC de Estados Unidos y Canadá. Fue diseñado para redefinir el formato original MARC del siglo XXI bajo el estándar ANSI Z39.2, el cual permite a los usuarios que utilizan distintos productos de software comunicarse entre sí e intercambiar información. Por otra parte, MARC 21 permite el uso de dos

conjuntos de caracteres: MARC-8 o Unicode codificado como UTF-8, admitiendo el uso de caracteres en todos los idiomas soportados por dichos estándares (8).
La estructura del formato MARC21 está compuesta por:

Líder: Se compone de una extensión fija de 24 caracteres.

Directorio: Enlista las etiquetas, a extensión de los campos y el inicio y fin de cada posición. Inicia en la posición 24, donde encuentran datos asociados con los que se registran en el campo variable.

Campos control: Registra datos que por su naturaleza son fijos en cuanto a su extensión.

Campos variables: Se encuentran datos bibliográficos descriptivos codificados en una de las etiquetas del formato MARC.

Los campos de datos se agrupan en bloques de acuerdo con el primer caracter de la etiqueta, la función de los datos dentro del registro está determinada generalmente por el bloque al que pertenece, como se muestra a continuación (9):

0XX- Información de control, números de identificación y clasificación, etc.
1XX- Asientos principales.
2XX- Títulos y párrafo del título (título, edición, pie de imprenta).
3XX- Descripción física, etc.
4XX- Menciones de serie.
5XX- Notas.
6XX- Campos de acceso temático.
7XX- Asientos secundarios diferentes a los de materias y series; campos ligados.
8XX- Asientos secundarios de series, existencias, etc.
9XX- Reservados para implementación local.

También se da significado a los campos a partir de los dos últimos caracteres de la etiqueta, aunque existen algunas excepciones.

X00 Nombres personales.

X10 Nombres corporativos.
X11 Nombres de reuniones.
X30 Títulos uniformes.
X40 Títulos bibliográficos.
X50 Términos temáticos.
X51 Nombres geográficos.

- **MARC-XML**

En 2002, la Biblioteca del Congreso desarrolló el esquema MARC-XML como una estructura alternativa de registro, que permite que los registros MARC sean representados en XML. Las bibliotecas usualmente exhiben sus registros en formato MARC-XML a través de un servicio web, con frecuencia siguiendo los estándares SRU (Search/Retrieve vía URL).

Entre los objetivos primarios de diseño de MARC-XML están:

- ✓ Flexibilidad y extensibilidad.
- ✓ Permitir que la conversión al formato MARC sea reversible y sin pérdidas de información.
- ✓ Facilitar la presentación de datos a través de hojas de estilo XML.
- ✓ Permitir actualizaciones de registros MARC y conversiones de datos por medio de transformaciones XML (10).

1.1.5. Migración de datos

Se denomina migración de datos, al proceso que tiene por objeto tanto la importación como la exportación de una determinada información almacenada en un sistema de bases de datos, para llevar a cabo su traspaso (11).

1.1.6. ¿Por qué la Migración de Datos?

Existen muchas circunstancias cuando una organización necesita migrar las aplicaciones o las bases de datos. Este proceso puede ser tan simple como una mejora a una nueva versión del sistema o que implique cambiar a una nueva base de datos o aplicación. Después de una fusión o de una adquisición, a

menudo se retiran las aplicaciones redundantes, pero los datos tienen que ser preservados en el sistema de supervivencia (12).

1.1.7. Procesos de migración de datos

La migración de datos está compuesta por 4 fases:

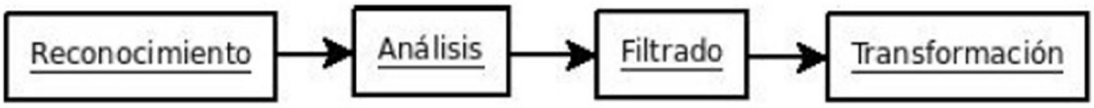


Figura 1 Fases de la migración de datos

1. **Reconocimiento:** Proceso en el cual se identifica la estructura de los datos en la fuente de datos.
2. **Análisis:** Proceso donde se define una estructura para almacenar de forma óptima los datos de la fuente, de modo que puedan realizarse los siguientes procesos de migración de forma eficiente, usualmente este proceso se mezcla con el de reconocimiento.
3. **Filtrado:** Este es un proceso opcional se realiza solo si se desea aplicar un filtro a los datos de alguna manera, ejemplo reducir su contenido tomando en cuenta cierto factor.
4. **Transformación:** En este proceso se realizan las transformaciones requeridas para que los datos tomen el formato deseado (13).

En el campo de las bibliotecas, la migración de datos depende completamente de la manera en que se desea representar la información. La delimitación de los datos y los campos de información son conceptos cruciales, de cara a facilitar la importación/exportación de un catálogo bibliográfico de un sistema de gestión bibliotecaria a otro.

Los delimitadores como su propio nombre indica, son marcas que permiten identificar los segmentos de datos o contenidos de un determinado registro. Por tanto, cualquier formato de exportación de datos que esté debidamente delimitado, representa una tabla de una base de datos y en consecuencia todos sus campos. De esta forma se puede obtener una copia íntegra y exacta de la información original, quedando reproducida tanto en su forma y estructura como en su contenido (11).

Para lograr trasladar no sólo la estructura sino un contenido o información de origen, existen múltiples formas de representar y delimitar los datos, los campos que los contienen y la estructura completa original. A menudo en los sistemas de automatización se desarrollan formatos propios, que plantean una codificación en la delimitación que posee en algunos casos gran complejidad. Esta situación limita la divulgación de materiales bibliográficos debido a que no se encuentra normalizado y es incompatible con otros sistemas de tratamiento de información. Para dar solución a este problema se han confeccionado formatos de migración o exportación de datos (11). A continuación se muestra la delimitación de campos e información en un formato de exportación:

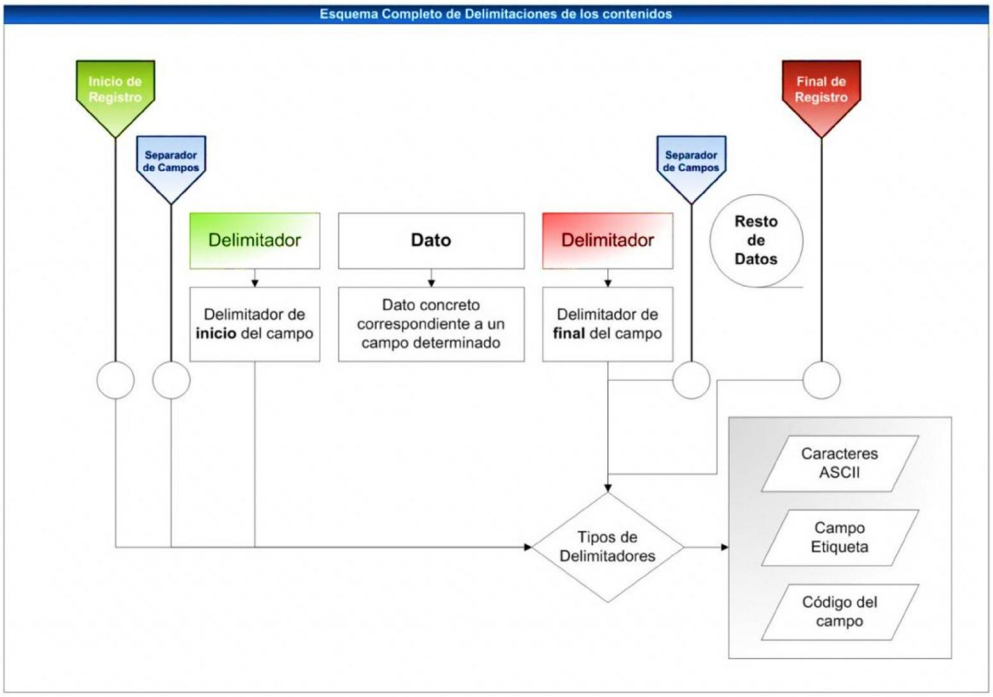


Figura 2 Delimitación de campos en formatos de exportación

Según se muestra en el esquema de delimitación de campos e información, es obligatorio establecer como mínimo dos delimitadores que correspondan al inicio y final de un dato o un campo. De esta forma, cualquier PARSER o analizador de datos, es capaz de seguir un patrón sencillo de la codificación que se ha establecido para el formato de exportación correspondiente. A su vez es necesario establecer un

delimitador que diferencia el inicio y el final de cada registro. Se debe tener en cuenta que un registro puede tener múltiples campos y múltiples datos, por lo que si se desean representar todos los datos de un registro bibliográfico, se requiere obligatoriamente diferenciar la separación entre los diferentes asientos de la tabla (11).

Por otra parte, todos los separadores responden al esquema presentado anteriormente, por lo que se observará siempre que la delimitación de datos y campos de un registro, mantienen un concepto de anidamiento singular como se muestra en la siguiente figura:

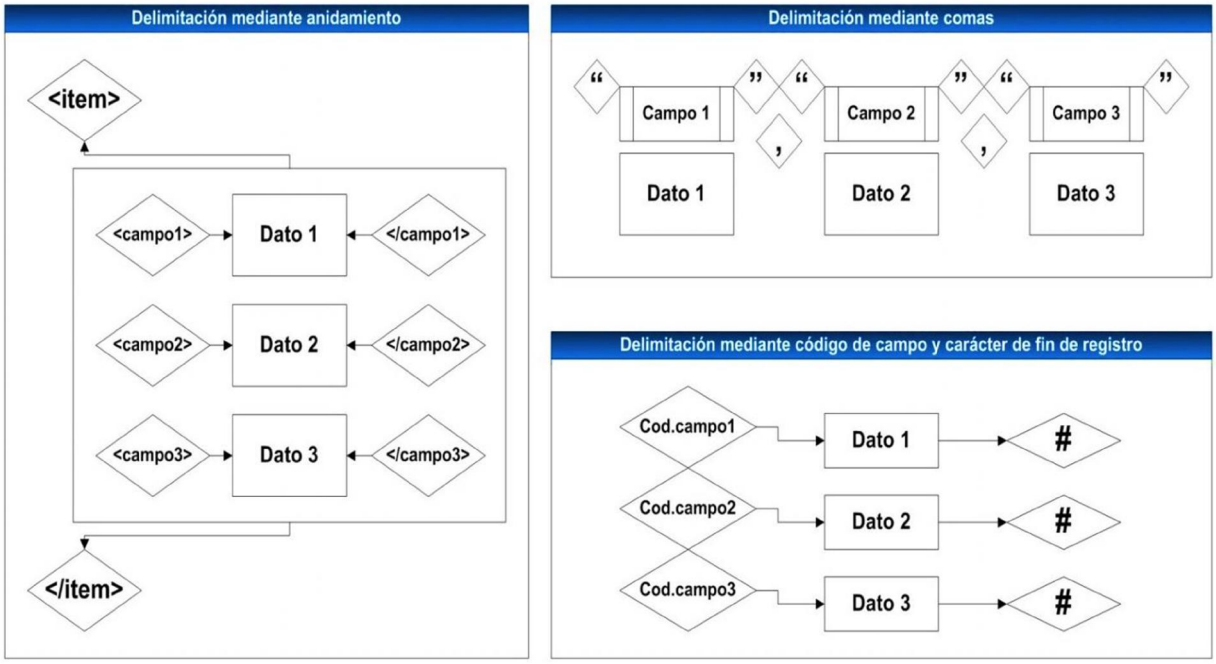


Figura 3 Tipos de delimitaciones de campos

A su vez, estos delimitadores se conforman mediante caracteres ASCII, términos a modo de etiquetas previamente establecidas, códigos que definen un campo (como en el caso de las etiquetas MARC), códigos alfanuméricos, mezcla de códigos y caracteres especiales ASCII, o combinaciones entre todos estos elementos, por lo que cualquier palabra o término inventado, puede ser un delimitador válido, siempre y cuando se emplee conforme a unas normas de descripción para un campo de una tabla. Lo

lógico es que en muchos casos los delimitadores sean los nombres de los campos de la tabla, de forma que se pueda identificar rápidamente la estructura original de la misma y encapsular la información (11).

Pero no siempre los formatos de exportación de información bibliográfica responden a estos esquemas. Un ejemplo de ello es el formato ISO2709 que emplea delimitadores iniciales diferentes a los delimitadores finales. En estos casos, el analizador de datos, define que el contenido de datos que encuentre en el espacio correspondiente entre la primera etiqueta de campo y la siguiente, corresponde a un dato de la primera etiqueta y así sucesivamente (11). Véase el siguiente esquema sobre delimitación de campos y datos en ISO2709-Unimarc.

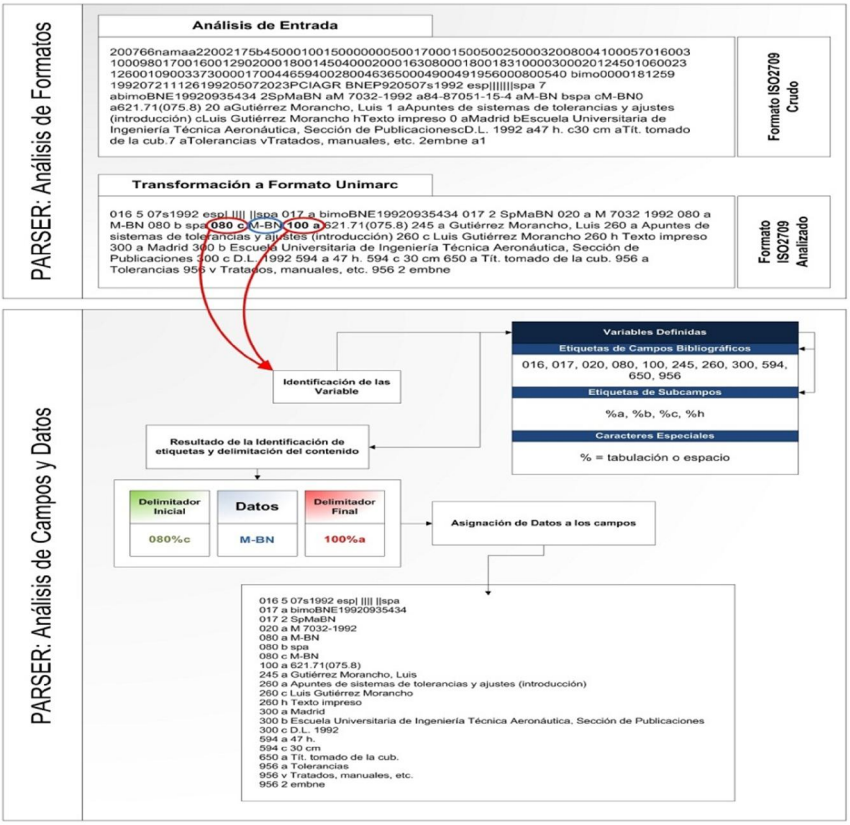


Figura 4 Delimitación de campos y datos en ISO2709-Unimarc

A este modelo especial, se añade el modelo de delimitación por anidamiento nodal, delimitación por caracteres especiales y caracteres especiales finales. En todos los casos es posible de igual forma establecer la distinción de los diferentes datos y sus correspondientes campos (11).

1.1.8. Formatos de migración más comunes

- **Formato CSV**

El formato de valores separados por comas permite representar todos los datos en forma de tabla, de forma que se adapta por completo a todas las tipologías de bases de datos y especificaciones de cada campo, con independencia de su naturaleza, ya sea numérico, fecha o texto.

Se caracteriza por utilizar las comas como separadores para diferenciar las columnas de la tabla. De esta forma el salto de línea representa una fila de datos nueva, diferente a la anterior. Otra de las reglas de construcción de un archivo CSV es la disposición de comillas dobles para separar columnas, siempre que en el valor textual de cada campo se incluyan comas como parte del contenido.

Para la importación y exportación de catálogos bibliográficos es muy adecuada, siempre que se definan correctamente las etiquetas de cada columna de forma adecuada y se generen todos los campos necesarios. Por esta razón se requiere de una exhaustiva descripción para reflejar todos los campos que el estándar ISO2709 establece (11).

- **Formato XML**

XML es uno de los mejores formatos para la exportación de la información de un catálogo bibliográfico. Destaca fundamentalmente por ser completamente adaptable a cualquier esquema de datos por complejo que pueda resultar. Esto se consigue gracias a que es un lenguaje de marcado extensible que depende de un Schema[7] o descripción del tipo de documento para su verificación. De esta forma se puede definir por completo las características de cada uno de los campos de descripción de un catálogo automatizado según las normas o reglas de catalogación establecidas. Un ejemplo de ello es el formato MARC XML, que permite adaptar sus registros a una serie de *items* (elementos) que reflejan todas las etiquetas, campos numéricos y subcampos correspondientes para la descripción bibliográfica. De esta forma se logra representar todo el etiquetado MARC atendiendo a una sintaxis y normas definida (11).

[7] Lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

Por otro lado, XML establece etiquetas de inicio y cierre de los contenidos, que permite definir de forma muy sencilla el nombre de los campos, sus atributos y valores. A diferencia de CSV, este permite el anidamiento de los registros favoreciendo la separación de cuantos asientos bibliográficos o registros sean necesarios, sin límites físicos en los archivos. Por este motivo, resulta más fiable porque reduce errores y pérdidas de información en las migraciones que se efectúen.

Las transformaciones XSLT[8] son una de las principales características de XML, que permiten convertir un documento XML a otro formato.

1.2. Análisis de herramientas relacionadas

1.2.1. Herramientas para la conversión de Ficheros Bibliográficos

- **IBEMARC-MARC 21**

A principios del año 2011 la Biblioteca Nacional de España (BNE) recomienda la migración de IBERMAC a MARC 21. Se crea un grupo de trabajo cuyo objetivo era llevar a cabo la migración para facilitar la labor a las bibliotecas y empresas implicadas y servir de cauce de la transmisión de la información entre la BNE, el Ministerio de Cultura y bibliotecas especializadas (14).

- **CDS/ISIS Utilities for Windows**

Esta herramienta brinda algunas utilidades como serían conversiones de datos CDS/ISIS en XML (ISIS2XML), además permite aplicar el proceso inverso (XML2ISIS). También permite la conversión de Isis a ASCII. (15).

- **MARConvert**

MARConvert es una herramienta que permite la conversión de los registros de entrada o salida de MARC21, UNIMARC, MARC-XML o formatos de autoridad. Además permite convertir registros MARC a otros formatos como:

  - ASCII.
  - Formatos de marcado como XML o HTML.
  - Tablas o bases de datos relacionales.

[8] Extensible Stylesheet Language Transformations es un lenguaje declarativo basado en XML

- De MARC21 a UNIMARC o inversa.
- De MARC21 a MARCXML o inversa.

1.2.2. Herramientas de migración de datos

- **PDI[9]-Kettle**

Kettle es un proyecto de código abierto que incluye un conjunto de herramientas para realizar ETL[10]. Forma parte de la suite de inteligencia de negocios de Pentaho. Entre sus aplicaciones más utilizadas se encuentra Spoon, una herramienta gráfica que permite transformaciones ETL, conectándose a nuevos orígenes de datos y transformarlos para cargarlos dentro de su estructura del almacén de datos. Permite la conexión a un gran número de bases de datos como Oracle, MySQL, PostgreSQL y SQLite (16).

- **Oracle Data Integrator (ODI)**

Esta herramienta presenta un esquema diferente del proceso de ETL, ya que está concebida como una herramienta de E-LT, esto significa que los datos se extraen de la fuente, se cargan en el sistema destino y ahí son transformados. Esto permite que la herramienta sea utilizada no solamente para etapas de ETL, sino para mover información de un sistema a otro, ya sea en grandes volúmenes por lotes o sólo unos pocos registros en tiempo real, habilitándola también como una herramienta de integración entre plataformas. De igual manera, cuenta con un componente para mejorar la calidad de la información que es cargada del sistema fuente al destino. Además puede utilizar diferentes motores de base de datos como sistema destino como el directorio LDAP, Oracle eBusiness Suite, Generic SQL, Microsoft Access, Oracle Database, Oracle Data Quality for Data Integrator y Teradata (17).

1.3. Ambiente de desarrollo

Con el objetivo de lograr el producto antes mencionado, se han realizado estudios sobre las herramientas y metodologías de desarrollo a utilizar basándose siempre en las facilidades que ofrecen para el ambiente

[9] Siglas en inglés Pentaho Data Integration

[10] Siglas en inglés de Extraer, Transformar y Cargar (Extract, Transform and Load). Es el proceso que permite mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos.

de desarrollo y las exigencias del cliente. Seguidamente se exponen características y ventajas de las mismas para poder tener un mayor conocimiento de sus prestaciones y valorar su utilidad.

1.3.1. Metodologías de desarrollo de software

Una buena selección de la metodología que se desea emplear favorece la calidad del software deseado, pues de ahí se deriva el papel que deben desempeñar cada miembro dentro del equipo de desarrollo y qué actividades tiene que cumplir cada uno de ellos. Además se definen qué artefactos deben ser creados y se pormenoriza cada detalle de la información del producto que se alcance como resultado de toda la actividad realizada. Estas metodologías se dividen en dos grandes grupos: ágiles y robustas. Actualmente, se cuenta con técnicas que teniendo en cuenta las características de cada software, permiten realizar una selección más acertada de la metodología a utilizar.

Para la selección del tipo de metodología a emplear se utilizó el *método Boehm y Turner.*

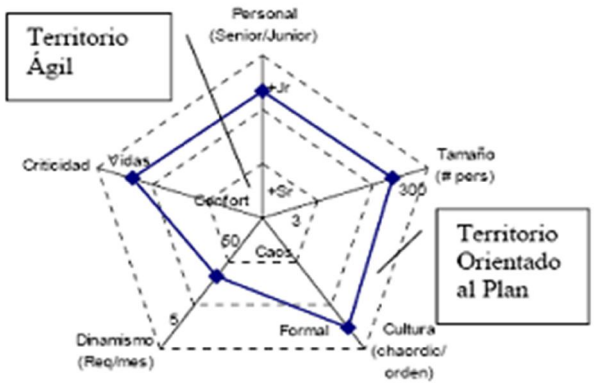


Figura 5 Representación de la estrella de Boehm y Turner

Este plantea 5 criterios fundamentales mediante los que se puede valorar el proyecto:

- ✓ **Tamaño:** Este criterio se utiliza para representar el número de personas involucradas en el proyecto. Pueden tenerse en cuenta el nivel de complejidad que pueda presentarse en la comunicación entre los miembros del proyecto y los costos que pueden provocar cambios esperados.

- ✓ **Criticidad:** Se utiliza para evaluar la naturaleza del daño ocasionado por defectos que no hayan sido detectados al producto. Su evaluación puede ser cualitativa.
- ✓ **Dinamismo:** Representa la rapidez con la que pueden estar cambiando los requerimientos del proyecto.
- ✓ **Personal:** Representa la proporción del personal con experiencia alta, media y baja. Los métodos orientados al plan no se ven afectados negativamente por este factor pues no interesa el nivel de experiencia con la que cuenten los miembros del equipo.
- ✓ **Cultura:** Las organizaciones y las personas que relaciona el proyecto pueden depender de la confianza o de la relación contractual. Esto refleja el nivel de ceremonia necesario y aceptado: documentación, control, formalismo en las comunicaciones (18).

Al aplicar el método Boehm y Turner, arrojó como resultado que el tipo de metodología que se ajusta las características de la solución propuesta es la *ágil (Ver Anexo 1)*.

La filosofía de las metodologías ágiles consiste en dar mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Por tales motivos, se propone utilizar para la elaboración de la solución propuesta una metodología ágil. A continuación se exponen algunos ejemplos de las metodologías ágiles más usadas a nivel mundial y en la UCI.

- **eXtreme Programming(XP)**

La metodología Programación Extrema o Extreme Programming (XP), es conocida como metodología ágil, orientada al cliente y de iteraciones cortas. La base para el desarrollo del software que usa esta metodología son las llamadas Historias de Usuario, que son escritas por el cliente en las que describen escenarios sobre el funcionamiento del sistema. Estas Historias de Usuario junto a la arquitectura que se persigue, sirven de base para crear un “plan de entregas de software” entre el equipo de desarrollo y el cliente (19).

En la figura se muestran las fases de la metodología XP y sus respectivos flujos de trabajo.

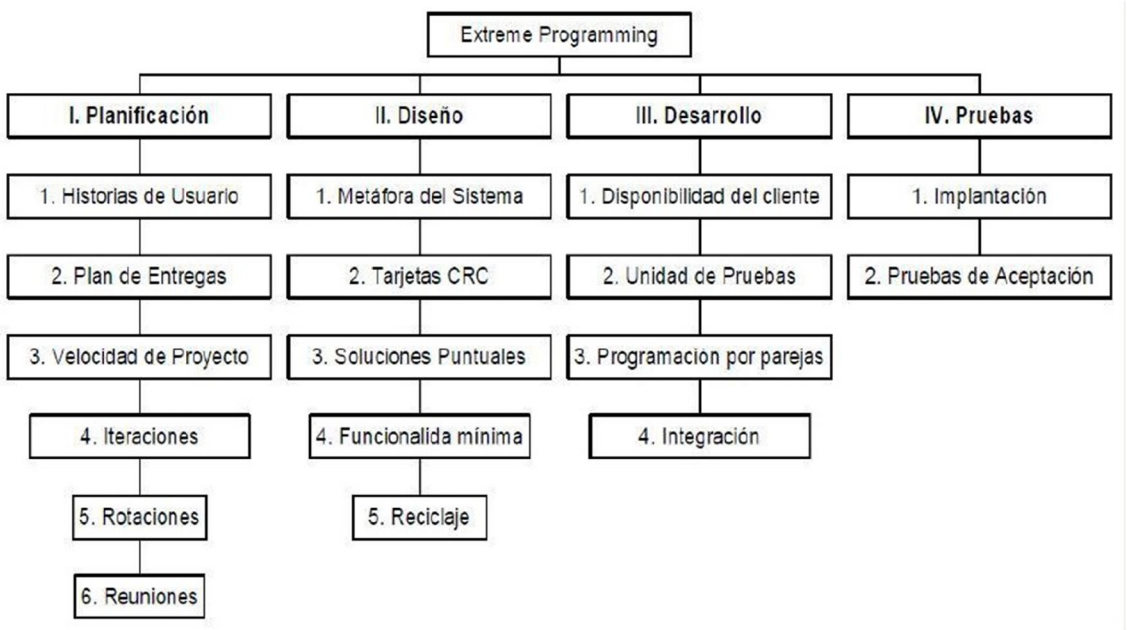


Figura 6 Fases y flujos de la metodología XP

XP propone que en el equipo de desarrollo se necesita la presencia del cliente que conozca detalladamente el negocio y que esté a disposición para cualquier duda que presenten los programadores. El cliente se mantiene todo el tiempo informado de cada paso de las actividades que se están desarrollando, y a medida que se dé la liberación de cualquier entregable se discutirá con el representante, y se repite la nueva iteración del software. La programación del software siempre se define en pareja con el objetivo principal de incrementar la calidad del software sin impactar el tiempo para cumplir con los requisitos del cliente. De esta forma, muchos errores son detectados durante el proceso de implementación, lo cual permite que los problemas de programación sean resueltos con rapidez. Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, ideas y soluciones de forma conjunta.

- **Desarrollo de Manejo por Rasgos (FDD)**

Es un proceso que se puede considerar a medio camino entre RUP y XP, aunque es más similar al último. Está pensado para proyectos de corto tiempo de duración (menos de un año) y propone iteraciones cortas

de aproximadamente dos semanas. Además, estas iteraciones se deciden en base a las funcionalidades definidas, divididas en pequeñas partes del software con significado para el cliente. Un proyecto con FDD se divide en cinco fases (19):

1. Desarrollo de un modelo general.
2. Construcción de la lista de funcionalidades.
3. Plan de releases en base a las funcionalidades a implementar.
4. Diseñar en base a las funcionalidades.
5. Implementar en base a las funcionalidades.

El trabajo se realiza en grupo aunque siempre habrá un responsable (arquitecto jefe o jefe de programadores, dependiendo de la fase en que se encuentre) con mayor experiencia. Las funcionalidades se dividen entre los subgrupos del equipo.

- **Proceso Unificado Ágil (AUP)**

AUP es una versión simplificada del Proceso Unificado de Rational (RUP) que describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles como: Desarrollo Dirigido por Pruebas (test driven development - TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad.

AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iniciales del proyecto. Además se desarrollan prototipos ejecutables durante la fase de elaboración del producto, donde se demuestra la validez de la arquitectura para los requisitos claves del producto y que determinan los riesgos técnicos.

Esta metodología establece un modelo más simple debido a que reúne en una única disciplina el Modelado de Negocio, Requisitos, Análisis y Diseño. El resto de las disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP.

Fundamentación de la metodología a emplear

Se procede a utilizar la metodología XP debido a que se adapta en gran medida, a las características de la solución propuesta y a las condiciones de trabajo. XP está concebida para ser utilizada dentro de proyectos pequeños. Haciendo uso de esta con la aceptación de nuevos requerimientos, el sistema debe cambiar y ampliar sus funcionalidades de forma que sea capaz de adaptarse a cada nueva situación. Además el equipo de trabajo se encuentra familiarizado con esta metodología lo cual es efectivo para reducir el tiempo de desarrollo de la aplicación.

1.3.2. Lenguaje de Modelado

- **Unified Modeling Language (UML)**

El Lenguaje Unificado de Modelado (UML) es una notación (esquemática en su mayor parte) con que se construyen sistemas por medio de conceptos orientados a objetos (20).UML es un lenguaje para construir modelos; no guía al desarrollador en la forma de realizar el análisis y diseño orientados a objetos ni le indica cual proceso de desarrollo adoptar. UML utiliza los diagramas gráficos para obtener distintos puntos de vista de un sistema:

- Diagramas de Casos de Uso.
- Diagramas de Clases.
- Diagramas de Comportamiento o Interacción.
- Diagramas de Implementación (21).

Características:

- Organiza el proceso de diseño.
- Reduce el período de desarrollo.
- Establece notaciones para realizar especificaciones visuales de procesos.
- Cuenta con reglas para combinar elementos gráficos.
- Los diagramas UML tienen como objetivo presentar diversas perspectivas de un sistema.

Se selecciona a UML debido a que el mismo facilita la comprensión de procesos específicos en la realización del software, aportando complementos visuales a los desarrolladores para modelar sistemas.

UML es adaptable a cualquier metodología de desarrollo de software, proporcionando artefactos para la comprensión y análisis de las diferentes fases involucradas en la elaboración de un software.

1.3.3. Herramienta de modelado a emplear

- **Visual Paradigm 5.0**

Visual Paradigm para UML (VP-UML) es una herramienta de diseño UML y herramienta CASE UML diseñada para ayudar al desarrollo de software. VP-UML soporta los principales estándares de la industria tales como UML, ofrece un conjunto completo de herramientas utilizadas por los equipos de desarrollo de software para la captura de requisitos, planificación de software, la planificación de controles, el modelado de clases, modelado de datos (22). Entre sus principales características sobresalen ser una aplicación multiplataforma, con copiosa información en línea, es un proyecto que se continúa actualizando, por lo tanto posee una comunidad de desarrolladores prestos para solucionar eficazmente cualquier inconveniente. Posibilita también la utilización de varios idiomas.
Debido a la facilidad y comodidad en el diseño de diagramas, provenientes de las diferentes fases del desarrollo del software se considera la utilización de Visual Paradigm para el diseño del Diagrama de Clases que proporcione una vista general de las clases que integran el sistema y las relaciones entre ellas.

1.3.4. Lenguaje de programación

- **C Sharp (C#)**

C# es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. Este es un lenguaje de programación simple, eficaz, con seguridad de tipos y orientada a objetos. Con sus diversas innovaciones, C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Constituye una evolución de los lenguajes C y C++. Utiliza muchas de las características de C++ en las áreas de instrucciones, expresiones y operadores. Presenta considerables mejoras e innovaciones en áreas como seguridad de tipos, control de versiones, eventos y recolección de elementos no utilizados

(liberación de memoria). C# proporciona acceso a los tipos de API[11] más comunes: .NET Framework, COM, Automatización y estilo C. Asimismo, admite el modo *unsafe*, en el que se pueden utilizar punteros para manipular memoria que no se encuentra bajo el control del recolector de elementos no utilizados. Además posibilita la interacción con componentes basados en tecnología Web, al ser compatible con XML (23).

- **Java**

Como lenguaje de programación posee características que lo hacen seguro, sencillo, simple, orientado a objeto, interpretado, portable y distribuido permitiendo altas prestaciones. Este lenguaje no depende del tipo de plataforma que emplea (24).

JDK[12] y JRE[13]: JDK es un paquete de software utilizado para desarrollar aplicaciones basadas en el lenguaje en Java. JDK necesita más espacio en el disco porque contiene el JRE junto con varias herramientas de desarrollo. Por otra parte, JRE es un conjunto de clases de API, compilador Java y archivos adicionales necesarios para escribir applets[14] y aplicaciones de Java. Es el encargado de ejecutar los programas de Java (25).

Librería Swing: Ofrece un gran poder para desarrollar interfaces gráficas de usuario (GUI) para applets y aplicaciones. Los componentes Swing están escritos en Java puro, lo cual ofrece ventajas significativas al permitir a los componentes Swing ser independientes del sistema de ventanas nativo, lo cual implica que pueden ejecutarse en cualquier sistema de ventanas (25).

Fundamentación del lenguaje de programación a emplear

Se seleccionó Java como lenguaje de programación debido a que es un lenguaje fácil de usar, multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Dicho lenguaje además es compilado, utiliza el paradigma orientado a objeto y en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento.

[11] Applications Programming Interface, o traducido al español Interfaz para la Programación de Aplicaciones.
[12] Java Development Kit
[13] Java Runtime Environment
[14] Es un tipo especial de programa Java que un navegador habilitado con tecnología Java puede descargar desde Internet y ejecutarlo.

Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, ofreciendo un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

1.3.5. Herramientas de desarrollo

- **Eclipse**

Entorno de desarrollo multilenguaje, de código abierto con una comunidad activa de desarrolladores de extensiones (plugins). Se utiliza para el trabajo con GWT[15] en el diseño de interfaces web enriquecidas haciendo uso de Java como lenguaje de programación. Eclipse permite el uso del plugin gwt-open layers para facilitar la integración del API de JavaScript Open Layers con GWT, unificando así el lenguaje de programación a utilizar. Este IDE tiene un alto acoplamiento con librerías como GWT EXT, y facilita el trabajo con herramientas como Open Layers[16].

- **NetBeans**

NetBeans IDE es un entorno de desarrollo integrado (IDE, por sus siglas en inglés), modular, de base estándar (normalizado), escrito en el lenguaje de programación Java. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (*framework*[17]) para compilar cualquier tipo de aplicación. Se ejecuta en Windows, Linux, MacOS X y Solaris y además es un producto de código abierto y gratuito sin restricciones de uso.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs[18] de NetBeans y un archivo especial (*manifest file*) que lo identifica

[15] **G**oogle **W**eb **T**oolkit (GWT) es una herramienta de desarrollo de aplicaciones web modernas. Permite al programador trabajar en un entorno ágil, de alto nivel, y despreocupándose de problemas como la incompatibilidad de navegadores o la realización de llamadas Ajax, para producir sistemas altamente interactivos, con todas las características que se espera en el contexto Web 2.0.

[16] Tecnología para desarrollar clientes Web.

[17] Del inglés marco de trabajo.

[18] Del inglés Applications Programming Interface o interfaz para la programación de aplicaciones.

como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que estos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. (26)

Fundamentación de la herramienta de desarrollo a emplear

Luego de realizar el análisis de las herramientas de desarrollo existentes, el equipo de desarrollo decidió utilizar NetBeans 7.3, ya que es una herramienta simple y fácil de usar. Posee una interfaz de usuario amigable y una excelente capacidad de autocompletado de código. Por otra parte, soporta el uso de librerías y flexibiliza el diseño de aplicaciones basadas en arquitecturas modulares. Además cuenta con mejor documentación que Eclipse ya que la mayoría de los documentos referentes a esta última son enlaces a *eclipse.org* que muchas veces son obsoletas.

Conclusiones parciales

En este capítulo fue elaborado el marco teórico orientado a las herramientas para la migración dinámica entre formatos bibliográficos. Para ello fue necesario realizar un estudio de los principales conceptos asociados a los SIGB y los formatos bibliográficos, para conocer los elementos técnicos que faciliten la comprensión del tema, así como el estudio de los formatos bibliográficos más utilizados. Además, se analizaron las fases del proceso de migración de datos y dentro de este los formatos más usados. Por otra parte, se realizó el estudio de herramientas encargadas de la conversión de ficheros bibliográficos que permiten realizar transformaciones a formatos bibliográficos pero enfocados en el mismo estándar. Sin embargo, estas soluciones no realizan transformaciones hacia formatos bibliográficos cuya estructura de almacenamiento es diferente para cada base de datos, como es el caso de los registros bibliográficos en formato MARC almacenados en MySQL o en CDS/ISIS. Por otra parte, las herramientas para la migración de datos analizadas, aunque permiten un gran número de migraciones de diversos orígenes y destinos, carecen de un driver de conexión a la base de datos CDS/ISIS. Finalmente, se realizó un estudio de las tecnologías, herramientas y metodología a emplear para la implementación de la solución propuesta.

Al finalizar el capítulo queda seleccionado UML como lenguaje de modelado y el lenguaje de programación Java para la implementación del software para la migración dinámica entre formatos bibliográficos por ser un lenguaje multiplataforma. En cuanto a la herramienta de modelado quedó seleccionado Visual Paradigm por las facilidades que ofrece para obtener una vista general de las clases y sus relaciones. Como IDE de desarrollo se decidió utilizar NetBeans en su versión 7.3. Fue seleccionada como metodología de desarrollo XP por su constante intercambio con el cliente y la hegemonía que brinda al equipo de desarrollo.

CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA

Introducción

Para garantizar que la implementación del software para la migración dinámica entre formatos bibliográficos sea exitosa y se alcance el producto esperado resulta obligatorio el análisis de los aspectos que intervienen en los procesos del objeto de automatización. Dicho análisis es realizado en este capítulo, atendiendo a los procesos relacionados al campo de acción, profundizándose en la fase de planificación propia de la metodología de desarrollo de software utilizada, donde se confeccionan las Historias de Usuario (HU).

2.1. Propuesta de solución

Dando respuesta a la problemática planteada y con el objetivo de cumplir con las especificaciones que el cliente solicita se presenta una propuesta de solución basada en una aplicación de escritorio que sirva para la migración dinámica entre formatos bibliográficos. Esta herramienta brindará la posibilidad de realizar la transformación de los registros bibliográficos, obteniéndolos de la base de datos bibliográfica de origen a través de un plugin encargado de extraer los datos, luego el sistema enviará los datos al conversor donde se realiza el proceso de transformación mediante una tabla de equivalencia donde el usuario tendrá la posibilidad de configurar las operaciones que desea realizar sobre los registros y estas serán validadas, verificando que cumpla con la estructura definida en el sistema. Finalmente se emplea el plugin de exportación para obtener el fichero que contiene los registros transformados en el formato necesario para ser utilizado por la base de datos destino.

Figura 7 Propuesta de solución

Definición y estructura de las operaciones de la tabla de equivalencia

La tabla de equivalencia permite definir cuáles son los campos y subcampos a migrar utilizando las operaciones de asignación, unión y división. Esta posee una estructura *Clave->Valor* donde la clave es el identificador de campo o subcampo y el valor es la operación y sus argumentos.
A continuación se muestra las operaciones definidas en la tabla de equivalencia para realizar la transformación de los registros bibliográficos:

- **Asignación**

  Esta operación permite a campos o subcampo asignarles el valor de otros campos o subcampos, como se muestra a continuación:

  c ### -> **campo**(###). Donde ### serían los identificadores del campo.

  c ###,X -> **campo**(###). Donde X sería el identificador del subcampo.

  c ### ->**subcampos**(###,Z). Donde Z sería el identificador del subcampo.

  c ###,Y->**subcampos**(###,X). Donde X, Y serían los identificadores del subcampo.

Ejemplo:

c 100 -> **campo**(024)

c 245,a -> **campo**(008)

c 001 -> **subcampo**(254,s)

c 124,a -> **subcampo** (650,c)

- **Unión:**

Esta operación permite a un campo o subcampo asignarle el valor de la unión de otros campos o subcampos separados por un caracter especificado, como se muestra a continuación:

c ### ->**union**(###;###,X)[C]. Donde C sería el caracter de unión.

c ###,C ->**union**(###;###,X)[C]. Donde C sería el caracter de unión.

Ejemplo:

c 008 ->**union**(014;056,b)[/]

c 910,i ->**union**(014;056,b)[/]

- **Split:**

Esta operación permite a un campo o subcampo asignarle el valor de la división de otro campo o subcampo por el caracter especificado y las posiciones a asignar, como se muestra a continuación:

c ### ->**division**(###)[C](p1,p2). Donde C es el caracter por el cual se desea dividir y p1, p2 son las posiciones que se desean asignar al identificador.

c ###,F ->**division**(###)[C](p1,p2). Donde F es el identificador del subcampo.

Ejemplo:

c 008 ->**division**(048)[/](1,2)

c 456,h ->**division**(048)[/](1,2)

El *dinamismo* de la solución propuesta consiste en permitir al usuario definir un esquema de los campos y subcampos que desea migrar y las operaciones que desea realizar sobre ellos. Además posibilita la implementación de plugins para la obtención de datos de un origen y su exportación a un destino.

Mediante esta propuesta de solución se desarrollará una aplicación que cumpla con los objetivos planteados, logrando la migración de los registros bibliográficos almacenados en sistemas de bases de datos diferentes.

2.2. Descripción del sistema

Para organizar los procesos fundamentales que abarca la propuesta de solución, fue necesario dividir el sistema en dos subsistemas fundamentales, los cuales serían los plugin encargados de la extracción de los datos del origen y los plugin encargados de la exportación hacia el fichero de destino. A continuación se presentan características de cada uno de los subsistemas:

- **Subsistema de extracción de registros bibliográficos**

Este subsistema responde a las primeras fases del proceso de migración: *Reconocimiento y Análisis*. Es iniciado a petición del usuario, autorizado para acceder al sistema, por lo tanto es un proceso que inicialmente se encuentra inactivo. Para lograr mejores resultados se definen los siguientes módulos:

> **Módulo de acceso a datos:** Es el encargado de gestionar el proceso de cargar los registros bibliográficos.
> **Módulo de extracción:** Se encarga del proceso de extracción de los registros bibliográficos, en dependencia del formato del archivo que contiene dichos registros.

- **Subsistema de transformación de metadatos**

El subsistema de transformación de metadatos es transparente al usuario, se inicia una vez que se obtienen los registros bibliográficos, proceso perteneciente al subsistema antes analizado. Este subsistema representa las fases del proceso de migración: *Filtrado y Transformación.*

Módulo de transformación: Su objetivo es transformar los datos extraídos de los registros bibliográficos a través de una tabla de equivalencia que corresponde con el formato bibliográfico al que se desea migrar.
Módulo de exportación: Tiene como propósito exportar los registros bibliográficos transformados al formato de fichero destino que es una extensión usada por los SIGB para leer dichos registros.

Módulo de control: Se encarga de verificar los resultados de la migración y genera reportes sobre posibles errores que ocurran al finalizar este proceso.

2.3. Funcionalidades del sistema

"Las características funcionales son características requeridas del sistema que expresan una capacidad de acción del mismo -una funcionalidad- generalmente expresada en una declaración en forma verbal." (27)

Para describir las funcionalidades que se espera que el sistema propuesto suministre se utilizan las Historias de Usuarios (HU).

2.3.1. Historias de Usuario

"Las historias de usuario son la forma en que se especifican en XP los requisitos del sistema. Estas se escriben desde la perspectiva del cliente aunque los desarrolladores pueden brindar ayuda en su identificación. El contenido de las mismas debe ser concreto y sencillo. " (28)

"Las HU también conducen el proceso de las pruebas de aceptación, que son empleados para verificar que las historias han sido implementadas correctamente. Existen diferencias entre las HU y la tradicional especificación de requisitos, siendo su principal diferencia el nivel de detalle. Las historias de usuario solamente proporcionan los detalles sobre la estimación de riesgo y cuánto tiempo conllevará la implementación de dicha historia." (28)

El cliente clasifica las HU teniendo en cuenta la *Escala Nominal de Prioridad en el Negocio* de la siguiente manera:

- ✓ **Alta:** se le otorga a las HU que constituyen funcionalidades de vital importancia en el desarrollo del proyecto.
- ✓ **Media:** se le otorga a las HU que para el cliente constituyen funcionalidades a tener en cuenta sin que tengan una afectación directa sobre el proyecto que se está desarrollando.
- ✓ **Baja:** se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de la estructura y que no tienen nada que ver con el proyecto en desarrollo.

Durante el análisis del objeto de automatización se identificaron un total de ocho (8) HU, de las cuales se describen 3 de ellas a continuación, para ver el resto remitirse al *Anexo 2*:

Historia de Usuario	
Número: 1	**Usuario:** Gestor de Migración
Nombre historia: Configurar plugin de entrada	
Prioridad en negocio: Alta	
Programador responsable: Andrés Selles González	
Descripción: Se introducen los parámetros para establecer la conexión a la base de datos donde se encuentran los registros. Los campos a llenar son: Pestaña de Configuración de la conexión: • Usuario • Contraseña • Base de datos	
Prototipo de interfaz de usuario:	

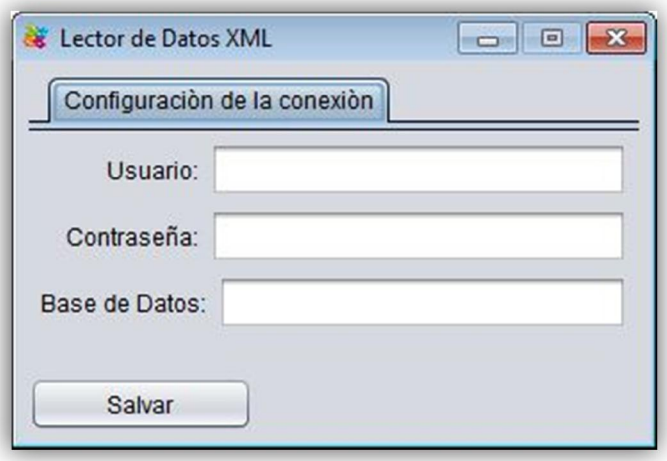


Tabla 1 Descripción de HU Configurar plugin de entrada

	Historia de Usuario
Número: 2	**Usuario:** Sistema
Nombre historia: Cargar registros bibliográficos	
Prioridad en negocio: Alta	
Programador responsable: Andrés Selles González	
Descripción: Accede a la base de datos donde se encuentran los registros bibliográficos y los extrae.	

Tabla 2 Descripción de HU Cargar registros bibliográficos

Luego que el cliente define las funcionalidades del sistema y describe las HU, estas son entregadas al programador, quien se encarga de estimar el riesgo de desarrollo de cada HU. De esta forma, las que son demasiado complejas, son entregadas al cliente nuevamente para que sean divididas y las que no se sabe cómo estimar, se les realizan pequeños programas de prueba *(spikes)*. A continuación se muestran las características no funcionalidades obtenidas de los *spikes*:

- **Software**

La computadora debe tener instalada la Máquina Virtual de Java (JVM).

- **Integridad**

Se implementará el uso de campos obligatorios y validaciones para garantizar la integridad de la información que se introduce por el usuario.

- **Diseño de Interfaz:**

El sistema debe tener una apariencia profesional y un diseño gráfico sencillo, con la utilización de las tonalidades de los colores azul, blanco y gris fundamentalmente. El texto incluido en la interfaz de la aplicación debe ser de tipo arial 11**.** Todos los textos y mensajes en pantalla aparecerán en idioma español. El sistema presentará los términos capitalizados, es decir, tendrán su primera letra en mayúsculas. Además en las interfaces se deben mostrar mensajes con información al usuario, los mensajes pueden ser: alertas, errores o información.

- **Portabilidad:**

El sistema podrá ser usado bajo el sistema operativo Linux o Windows.

2.4. Planificación

Luego de realizar la estimación de las HU se reúne el cliente y programadores y se realiza la clasificación de estas en función del riesgo teniendo en cuenta la *Escala Nominal de Riesgo en Desarrollo*:

- ✓ **Alta (1):** se otorga cuando para la implementación de la HU se considera la posible existencia de errores que lleven a la inoperatividad del código.
- ✓ **Media (2):** se otorga cuando pueden aparecer errores en la implementación de la HU que puedan retrasar la entrega de la versión.
- ✓ **Baja (3):** se otorga cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan perjuicios para el desarrollo del proyecto.

No. HU	Historia de Usuario	Riesgo en Desarrollo
1.	Configurar plugin de entrada	1
2.	Cargar registros bibliográficos	1
3.	Extraer registros bibliográfico	1
4.	Configurar tabla de equivalencia.	1
5.	Configurar plugin de salida	1
6.	Transformar registros bibliográficos.	1
7.	Exportar a formato destino	2
8.	Mostrar resultados de la migración.	3

Tabla 3 Riesgo en desarrollo por HU

2.4.1. Estimación de esfuerzo por Historia de Usuario

"Las estimaciones de esfuerzo asociado a la implementación de las HU se establecen utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, se mantiene un registro de la velocidad de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración." (29)

Para el desarrollo de la aplicación propuesta se tienen en cuenta situaciones relativas a HU tales como: aparición de nuevas, postergación en cuanto a la iteración en la cual se implementa, no finalización en la iteración planificada y modificación durante la iteración o una vez completada en una iteración previa, realizándose una estimación del esfuerzo para cada una de las HU identificadas, los resultados de este estudio se muestran en la siguiente tabla:

No. HU	Historia de Usuario	Puntos de estimación

1.	Configurar plugin de entrada	2
2.	Cargar registros bibliográficos	1
3.	Extraer registros bibliográfico	1
4.	Configurar tabla de equivalencia.	2
5.	Configurar plugin de salida	2
6.	Transformar registros bibliográficos.	1
7.	Exportar a formato destino	1
8.	Mostrar resultados de la migración.	1

Tabla 4 Estimación de esfuerzo por HU

Como parte del ciclo de vida del software se crea la planificación de duración de las iteraciones, la cual tiene como objetivo mostrar la duración de cada iteración y orden en que serán implementadas las HU en cada una de ellas. El desarrollo del sistema se dividió en tres (3) iteraciones, estas se definen de la siguiente forma:

Iteración 1

En la iteración 1 se implementan las HU correspondientes al subsistema de extracción de registros bibliográficos, es decir la 1, 2, 3 y 4, las mismas tienen una prioridad alta dada la complejidad del sistema en su conjunto, obteniéndose así la versión 0.1 del producto dándole la posibilidad al cliente de probar dichas funcionalidades, que son de carácter crítico para él.

Iteración 2

El objetivo de esta iteración es la implementación de las HU 5 y 6 encargadas de la transformación de los metadatos extraídos de los registros bibliográficos. Una vez concluida se contará con una nueva versión de prueba, la cual será mostrada al cliente con el objetivo de retroalimentar al grupo de trabajo.

Iteración 3

En la tercera iteración implementan las HU que terminan con los objetivos propuestos para el sistema, es decir las HU 7 y 8. Al finalizar la misma se contará con la versión 1.0 del producto final, por tanto el sistema será probado en el SIGB (KOHA) para valorar su usabilidad.

A continuación se muestra el orden de desarrollo de las HU en cada iteración en dependencia de la prioridad que tengan y el tiempo de duración de las mismas:

Iteración	Orden de las HU a implementar	Duración (Semanas)
Iteración 1	1. Configurar plugin de entrada	4
	2. Cargar registros bibliográficos	
	3. Extraer registros bibliográfico	
	4. Configurar tabla de equivalencia.	
Iteración 2	1. Configurar plugin de salida	4
	2. Transformar registros bibliográficos.	
Iteración 3	1. Exportar a formato destino	4
	2. Mostrar resultados de la migración.	

Tabla 5 Plan de duración de las iteraciones

2.4.2. Plan de entregas

"Las HU servirán para crear el plan estimado de entregas. Este plan se usa para crear los planes de iteración para cada iteración. Con cada HU previamente evaluada en tiempo de desarrollo ideal, el cliente las agrupará en orden de importancia." (30)

Para trazar el plan de entregas del software para la migración dinámica entre formatos bibliográficos se realiza en función de dos parámetros: tiempo de desarrollo ideal y grado de importancia para el cliente.

Por otra parte las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración.

Seguidamente se presenta el plan de entregas estimado para la fase de implementación, como producto del presente plan se realizarán versiones entregables del sistema al finalizar cada una de las iteraciones, teniendo en cuenta que la entrega de estas versiones del *software* ocurre muy temprano y en intervalos muy cortos para maximizar la interacción con el usuario.

Sistema	Fin iteración 1 12/4/2013	Fin iteración 2 10/5/2013	Fin iteración 3 4/6/2013
Software para la migración dinámica entre formatos bibliográficos	Versión- 0.1	Versión-0.2	Versión-1.0

Tabla 6 Plan de entregas

2.4.3. Roles del sistema

La información del sistema sólo puede ser accedida y modificada por las personas autorizadas y de la forma autorizada, lo que significa que se definen roles para acceder al sistema, estos roles deben asignarse adecuadamente a los diferentes tipos de personas, según sus capacidades y puestos de cada una de ellas. La interacción directa con el sistema lo realizará:

Persona que interactúa con el sistema	Descripción
Gestor de migración	Posee los permisos para acceder a la base de datos bibliográfica y modificarla a través del proceso de

	migración.

Tabla 7 Roles del sistema

Conclusiones parciales

El análisis previo indica que con un mejor entendimiento del flujo actual de las actividades y el objeto de automatización se crean las bases para la comprensión de la solución a implementar. Durante este capítulo se realizó la descripción de la propuesta de solución de las HU definidas por el cliente, como paso fundamental en el ciclo de desarrollo de la aplicación así como las características no funcionales que este debe tener. Además se determinó el riesgo y tiempo de desarrollo de cada una de ellas por parte de los desarrolladores para lograr la implementación de la solución a través de tres iteraciones. De esta forma se detectarán las posibles inconsistencias al final de cada iteración y poder mitigarlas a tiempo.

CAPÍTULO 3. DISEÑO, DESARROLLO Y PRUEBA DEL SISTEMA

Introducción

Cada paso durante el ciclo de desarrollo del software cumple un importante papel para obtener el producto final. Como resultado del diseño se procede a la implementación del sistema en términos de componentes: ejecutables, ficheros de código fuente, *scripts*, entre otros. Tiene como objetivo desarrollar la arquitectura y el sistema como un todo, así como definir la organización del código.

Para lograr una correcta implementación, es necesario que la solución propuesta satisfaga las expectativas del usuario final, asegurando que sea operacional o que funcione de acuerdo a los requerimientos. Derivándose entonces la incorporación de pruebas al sistema, actividad para garantizar la calidad del *software*. En el presente capítulo, se abordan las fases de diseño, desarrollo y prueba.

3.1. Diseño

El diseño, durante el ciclo de vida del software es “el proceso de aplicar distintas técnicas y principios con el propósito de definir un producto con los suficientes detalles como para permitir su realización física. Con el diseño se pretende construir un sistema que se ajuste a las limitaciones impuestas por el medio de destino y respete requisitos sobre forma, rendimiento, utilización de recursos, coste, etc.” (31). Para producir la representación técnica del *software* a desarrollar se plantea el diseño evolutivo, es decir, se realiza a medida que el sistema evoluciona, formando parte de la programación. “En el diseño evolutivo XP aboga por hacer la cosa más simple que pueda funcionar y que no lleve a cabo todo aquello que no sea necesario” (28).

En el caso de la aplicación para la migración dinámica entre formatos bibliográficos, se basa en un diseño simple y claro, es decir, la simplicidad es la llave pues cuesta menos tiempo de implementar un diseño sencillo que uno complejo. En el presente epígrafe se exponen los elementos relevantes para el diseño del sistema.

3.1.1. Arquitectura de la solución

"Desde los sistemas más pequeños hasta los más grandes poseen una estructura y un comportamiento que los hace clasificables según su arquitectura, la cual establece el funcionamiento e interacción entre las partes del *software*." (32). La propuesta de solución está basada en una arquitectura *híbrida*, compuesta por la arquitectura N-Capas para el sistema y 3 Capas para la implementación de los plugin. Esto permite la separación de responsabilidades en cada una de las capas y la reutilización, escalabilidad y facilidad para el desarrollo del sistema. Dichas capas están bien delimitadas una de la otra, una capa superior interactúa con la inferior mediante interfaces que definen las funcionalidades que la misma debe brindar.

- **Patrón Arquitectónico: N-Capas**

Este patrón define cómo organizar el modelo de diseño a través de capas, que pueden estar físicamente distribuidas, lo que significa que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Además, dicho patrón simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

- **Descripción de la arquitectura del sistema**

La arquitectura del sistema posee 2 capas lógicas:

Presentación: Esta capa se encarga de proveer una interfaz entre el sistema y el usuario. Básicamente, se responsabiliza de que se le comunique información al usuario por parte del sistema y viceversa, manteniendo una comunicación exclusiva con la capa de negocio que es descrita a continuación.

Lógica de negocio: Es la capa que contiene los procesos a realizar con la información recibida desde la capa de presentación, las peticiones que el usuario ha realizado y es responsable del envío de las respuestas adecuadas a la capa de presentación.

Esta última capa es adaptada a la solución propuesta, debido a que el proceso de migración depende en gran medida de la lógica implementada en los plugins como se muestra a continuación:

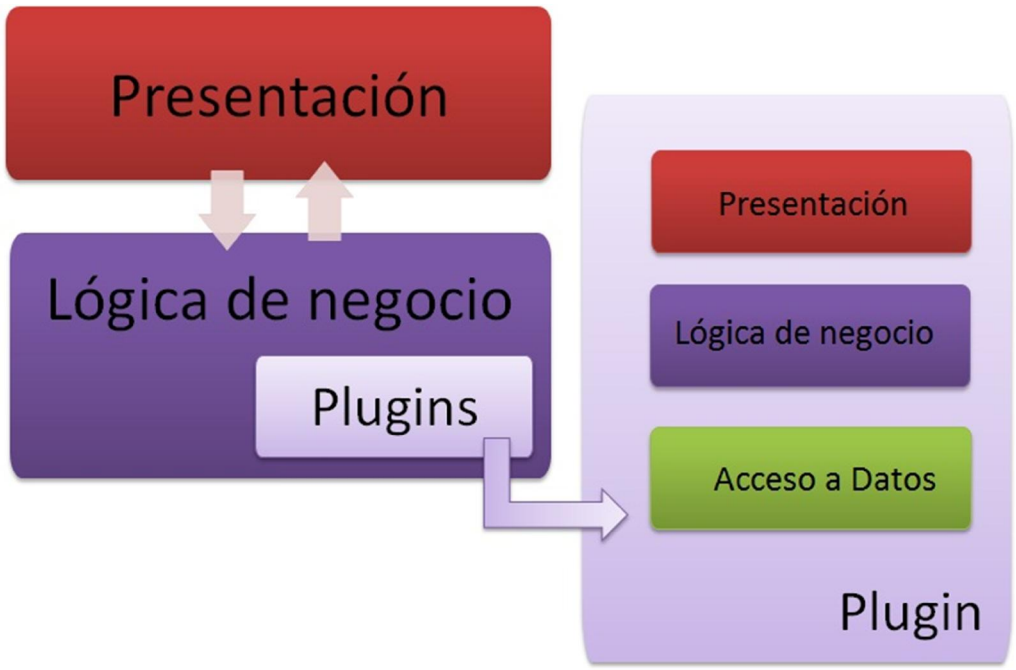


Figura 8 Representación de arquitectura del sistema

Capa de Presentación:

- ✓ *Principal:* Brinda acceso a las funcionalidades principales de la aplicación. Permite al usuario añadir plugin a la aplicación desde un directorio externo y seleccionar los plugin para importar los registros desde el origen y exportarlos al destino luego de ser transformados.
- ✓ *EquivalenciaForm:* El usuario introduce las relaciones de equivalencia y las envía para ser procesadas en la capa de negocio.

Capa de Lógica de Negocio:

- ✓ *Conversor:* Se realizan las funciones seleccionadas por el usuario en la interfaz principal.
  - *anadirPlugin ():* Copia los plugin hacia el directorio *Plugin* de la aplicación.
  - *transformarDatos ():* Inicia el proceso de transformación a través de los plugin seleccionados.

- ✓ *Equivalencia*: Obtiene las configuraciones realizadas por el usuario en la interfaz *EquivalenciaForm* y valida la estructura de la tabla.

- **Descripción de la arquitectura de plugin**

La utilización del patrón arquitectónico 3 Capas se puede observar en la implementación de los plugin. A esta arquitectura se suma la capa de Acceso a datos encargada de obtener de la base de datos los registros bibliográficos a los cuales se le realiza el proceso de transformación.

Capa de Presentación:

- ✓ *LectorDatos*: Solicita al usuario los datos necesarios para la extracción de los registros bibliográficos.
- ✓ *GuardarDatos*: Solicita al usuario los datos necesarios para guardar los datos que serán transformados.

Capa de Lógica de Negocio:

- ✓ *PluginLoaderInterface*: Clase interfaz de la cual implementarán todos los plugin de la entrada de datos.
- ✓ *PluginExportarInterface:* Clase interfaz de la cual implementarán todos los plugin de la salida de datos.
- ✓ *PluginLoaderImpXML*: Implementación de la interfaz *PluginLoaderInterface,* implementando las funcionalidades del plugin para la extracción de los datos.
- ✓ *PluginExportarImplABCD:* Implementa la interfaz *PluginExportarInterface*, realizando las funcionalidades necesarias para exportar los datos.
- ✓ *ExtractorXML*: Encargado de realizar las operaciones de extracción de los registros bibliográficos.
- ✓ *TranformadorDatos*: Realiza el proceso de transformación de los registros bibliográficos teniendo en cuenta las relaciones de equivalencia y los salva.

Capa de Acceso a Datos:

Conexion: Establece la conexión con la base de datos donde se encuentran los registros bibliográficos.

- **Pautas para la creación de un plugin**

Para la implementación de los plugin se crean dos tipos, unos son los encargados de la extracción de los datos del formato de origen y los otros se encargan de exportar los datos al formato de destino utilizando las transformaciones formuladas en la tabla de equivalencia. La máquina virtual de java carga todas las librerías que se encuentran en el classpath[19] al iniciar la aplicación. Por tanto, para añadirle nuevos plugins a la aplicación mientras está en ejecución es necesario modificar el ClassPath. Para esto se crean dos clases: *ModificadorClassPath.java* y *CargadorPlugins.java*, la primera es la encargada de adicionar la dirección de las librerías dentro de la clase *URLClassLoader.java*, para que estén disponibles para su uso. La segunda es responsable de buscar dentro de la carpeta de los plugins todos los archivos de extensión .jar, obtener las direcciones de los mismos y pasarle esa dirección al *ModificadorClassPath.java*.

Plugins para extracción de los datos del origen:

Los plugins para la extracción de los datos deben ser creados como un proyecto independiente añadiendo como librería la referencia al proyecto PluginLoaderInterface. Este proyecto independiente deberá tener un nombre de la siguiente forma *PluginLoaderImplXML* seguido de una palabra o palabras que lo identifiquen. La clase que implementará la interfaz deberá tener el mismo nombre del proyecto e implementar los métodos de la interfaz PluginLoaderInterface. En el método *void openFormPrincipal* () se deberá llamar al formulario principal del plugin desde el cual se realizarán las operaciones necesarias para extraer los datos.

Para que la aplicación que utiliza los plugin los reconozca, se debe crear un archivo en blanco dentro del plugin en el directorio META-INF\services que tiene como nombre la dirección donde se encuentra la clase interfaz, quedando de la siguiente forma: pluginloaderinterface.plugins.PluginLoaderInterface, y dentro de este archivo se debe especificar la dirección donde se encuentran las clases que implementan esta

[19] Ruta de la cual la JRE busca las clases u otros archivos de recursos. Informa a las herramientas y aplicaciones de la JDK donde encontrar clases de terceros y creadas por el usuario, es decir, las clases que no son extensiones de java o parte de la plataforma de java.

interfaz: pluginloaderimpl.plugins.PluginLoaderImplXML. Este proceso se realiza para cada proyecto o implementación de plugins. En caso de que en un mismo proyecto se implemente más de un plugin, se deberá especificar en una nueva línea del fichero creado en META-INF\services la dirección de esta implementación. En cada implementación de los plugins se extraerán los datos del origen seleccionando la dirección donde se encuentran los mismos y serán almacenados en variables de tipo *Registro.java*, *Campo.java*, *SubCampo.java*. Se brindan las clases antes mencionadas para almacenar los datos de los registros, campos y subcampos en el proyecto ClasesRegistro que deberá ser añadido como referencia a las librerías del plugin que se implementará para poder usarlas. Además en este último proyecto se brindan otras clases auxiliares como *AdministradorArchivo.java* que permite realizar operaciones con archivos, como sería la copia de los plugins hacia la carpeta "plugins", donde se guardarán las implementaciones de entrada y salida.

Plugins para la exportación al fichero de destino:

Se crea un proyecto *PluginExportarInterface* en el cual se crea una clase interfaz *PluginExportarInterface.java*, que deberán implementar los plugins de salida que se deseen crear, en esta estarán definidos todos los métodos que implementarán estos plugins.

Los plugins de salida de datos a crear deberán ser creados como un proyecto independiente añadiendo como librería la referencia al proyecto *PluginExportarInterface*. Este proyecto independiente deberá tener un nombre de la siguiente forma *PluginExportarImpl* seguido de una palabra o palabras que lo identifiquen. La clase que implementará la interfaz deberá tener el mismo nombre del proyecto e implementar los métodos de la interfaz *PluginExportarInterface*. Para que la aplicación reconozca un plugin se debe crear un archivo en blanco dentro del directorio META-INF\services que tiene como nombre la dirección donde se encuentra la clase interfaz, para esta interfaz quedaría de la siguiente forma: pluginexportarinterface.plugins.ExportarInterface, y dentro de este archivo especificarle la dirección donde se encuentran las clases que implementan esta interfaz: pluginexportarimpl.plugins.PluginExportarImplABCD. Este proceso se realiza para cada proyecto o implementación de plugins. En caso de que en un mismo proyecto se implementen más de un plugin, se deberá especificar en una nueva línea del fichero creado en META-INF\services la dirección de esta implementación. En cada implementación de los plugins se transformarán los datos almacenados en variables de tipo *Registro.java, Campo.java, SubCampo.java.* Se brindan las clases antes mencionadas

para almacenar los datos de los registros, campos y subcampos en el proyecto *ClasesRegistro* que deberá ser añadido como referencia a las librerías del plugin que se implementará para poder usarlas.

3.1.2. Patrones de diseño

El desarrollo de sistemas informáticos motiva la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces pues facilitan la reusabilidad, extensibilidad y mantenimiento, problemas muy complejos, pero resueltos con anterioridad. Por esta razón uno de los pasos a tener en cuenta cuando se decide construir un proyecto de *software* es identificar que patrones pueden ser utilizados.

Christopher Alexander, arquitecto destacado, conocido por uno de sus libros El lenguaje de patrones (*A Pattern Language),* expresa que "cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma" (33)

Para estandarizar el modo en que se realiza el diseño en la aplicación se han tenido en cuenta patrones que logran el equilibrio necesario entre flexibilidad y rendimiento, los cuales se explican a continuación:

- **Patrones GOF[20]**

Los patrones de diseño del grupo de GOF se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Abstract Factory: Proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. Este patrón es usado en un sistema que deba ser independiente de cómo se crean, componen y representan sus productos (34). Un ejemplo de su uso se aprecia en la clase *PluginLoaderInterface,* que se encarga de definir la interfaces que se encuentran implementadas en clases que están relacionadas o dependen entre sí, la clase *PluginLoaderImplXML* es una implementación de esta interfaz.

[20] Del inglés *Gang of Four* o Banda de Cuatro.

```
package pluginloaderinterface.plugins;

public interface PluginLoaderInterface {

    void openFormPrincipal();

    String getNombreAMostrar();
}
```

```
package pluginloaderimpl.plugins;

import forms.cargadatos.LectorDatos;
import pluginloaderinterface.plugins.PluginLoaderInterface;

public class PluginLoaderImplXML implements PluginLoaderInterface {

    @Override
    public void openFormPrincipal() {
        LectorDatos lectorDatos = new LectorDatos();
        lectorDatos.setVisible(true);
    }

    @Override
    public String getNombreAMostrar() {
        return "Plugin XML";
    }
}
```

Singleton: Garantiza que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma. Este patrón es utilizado cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes. Se evidencia su utilización en el formulario *Principal*, donde se crea un objeto de la clase controladora *Conversor* y al crear una instancia del formulario *EquivalenciaForm* en el *Principal*, solicita el objeto de la controladora garantizando que exista una única instancia de la misma.

- **Patrones GRASP**[21]

"Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable." (34)
El uso de estos patrones está totalmente ligado a cada componente desarrollado en el sistema, donde cada uno de ellos posee sólo las funcionalidades acorde a las particularidades que lo caracterizan:

[21] Del inglés *General Responsability Assignment Software Patterns o* Patrones Generales de *Software* para asignación de Responsabilidades.

Creador: "Se aplica para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello." (34) El uso de este patrón permite crear las dependencias mínimas necesarias entre las clases, lo que favorece al mantenimiento del sistema y ofrece mejores oportunidades de reutilización, evidenciándose su uso en la clase *Principal*:

```
private void jMenuItem8ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileFilter(new FileNameExtensionFilter(".jar", "jar"));
    int accion = fileChooser.showOpenDialog(jMenu1);
    if (accion == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser.getSelectedFile();
        File file1 = new File("plugins");
        FileManager fileManager = new FileManager();
        fileManager.copiarPlugins(file, file1);
        cargarPluginsMethod();
    }
}
```

Experto: "Se aplica para la asignación de responsabilidades a las clases de forma tal que las mismas contengan la información necesaria para poder ejecutar una acción específica. El uso de este patrón permitirá a los objetos valerse de su propia información para hacer lo que se les pide, favorece la existencia de mínimas relaciones entre las clases, lo que permite contar con un sistema robusto y fácil de mantener." (34) En la clase *ModificadorClassPath* se evidencia la utilización de este patrón:

```
public class ModificadorClassPath {

    private static final String METODO_ADD_URL = "addURL";
    private static final Class[] PARAMETRO_METODO = new Class[](URL.class);
    private final URLClassLoader loader;
    private final Method metodoAdd;

    public ModificadorClassPath() throws NoSuchMethodException {
        loader = (URLClassLoader) ClassLoader.getSystemClassLoader();
        metodoAdd = URLClassLoader.class.getDeclaredMethod(METODO_ADD_URL, PARAMETRO_METODO);
        metodoAdd.setAccessible(true);
    }
```

Bajo acoplamiento: "El acoplamiento mide la fuerza con que una clase está conectada a otra, de esta forma una clase con bajo acoplamiento debe tener un número mínimo de dependencia con otras clases.

Las diferentes clases controladoras sólo dependen de un único controlador frontal para realizar sus funcionalidades.” (34) Este patrón se tuvo presente debido a la importancia que se le atribuye a realizar un diseño de clases independientes que puedan soportar los cambios de una manera fácil y que a su vez permitan la reutilización. Las clases donde se observa la utilización de dicho patrón son *CargadorPlugins, LectorDatos* y *FileManager.*

Alta Cohesión: “Se aplica para realizar un diseño que evite contener clases con un alto grado de abstracción, que asuman responsabilidades que podían haber delegado a otros objetos o que tengan responsabilidades muy complejas.” (34) Se tiene la clase *Conversor* que se encarga de ejecutar acciones de acuerdo a las peticiones que le llegan y la clase *PluginLoaderInterface* que define las interfaces que se implementan en otra clase, de forma tal que se elimina la sobrecarga de funcionalidades en las clases controladoras.

La arquitectura base y el diseño flexible a través del correcto uso de patrones de diseño en la generación de los artefactos vitales para el desarrollo del sistema posibilitan crear una entrada apropiada como punto de partida a las actividades de implementación, con el objetivo de lograr una mayor calidad del producto y el agrado del cliente.

3.1.3. Tarjetas CRC

“La metodología XP para el diseño de las aplicaciones no requiere la representación del sistema mediante diagramas de clases. En su lugar se usan variantes como las tarjetas CRC[22]. Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Las tarjetas CRC permiten que el equipo completo contribuya en la tarea del diseño. Una tarjeta CRC representa un objeto.” (35)

En las tarjetas CRC una **responsabilidad** es cualquier cosa que la clase sabe o hace. Los **colaboradores** son aquellas clases que se requieren para que una clase reciba la información necesaria para completar una responsabilidad.

[22] Cargo o Clase, Responsabilidad y Colaboración.

Sin embargo el equipo consideró necesaria la modelación de un diagrama de clases del diseño que describa la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. Para la elaboración de este diagrama fueron utilizadas las técnicas de abstracción en paquetes y en colores para agrupar las clases que pertenecen a cada capa de la arquitectura y diferenciarlas mediante un color, brindando al programador mayor organización de las clases del sistema.

Sobre el modelado UML en colores, Peter Coad, Eric Lefevre y Jeff De Luca realizaron investigaciones demostrando que es una técnica que nos permite identificar a simple vista, los distintos tipos de elementos que existan en un sistema, y además nos permite saber qué tipos de elementos son.

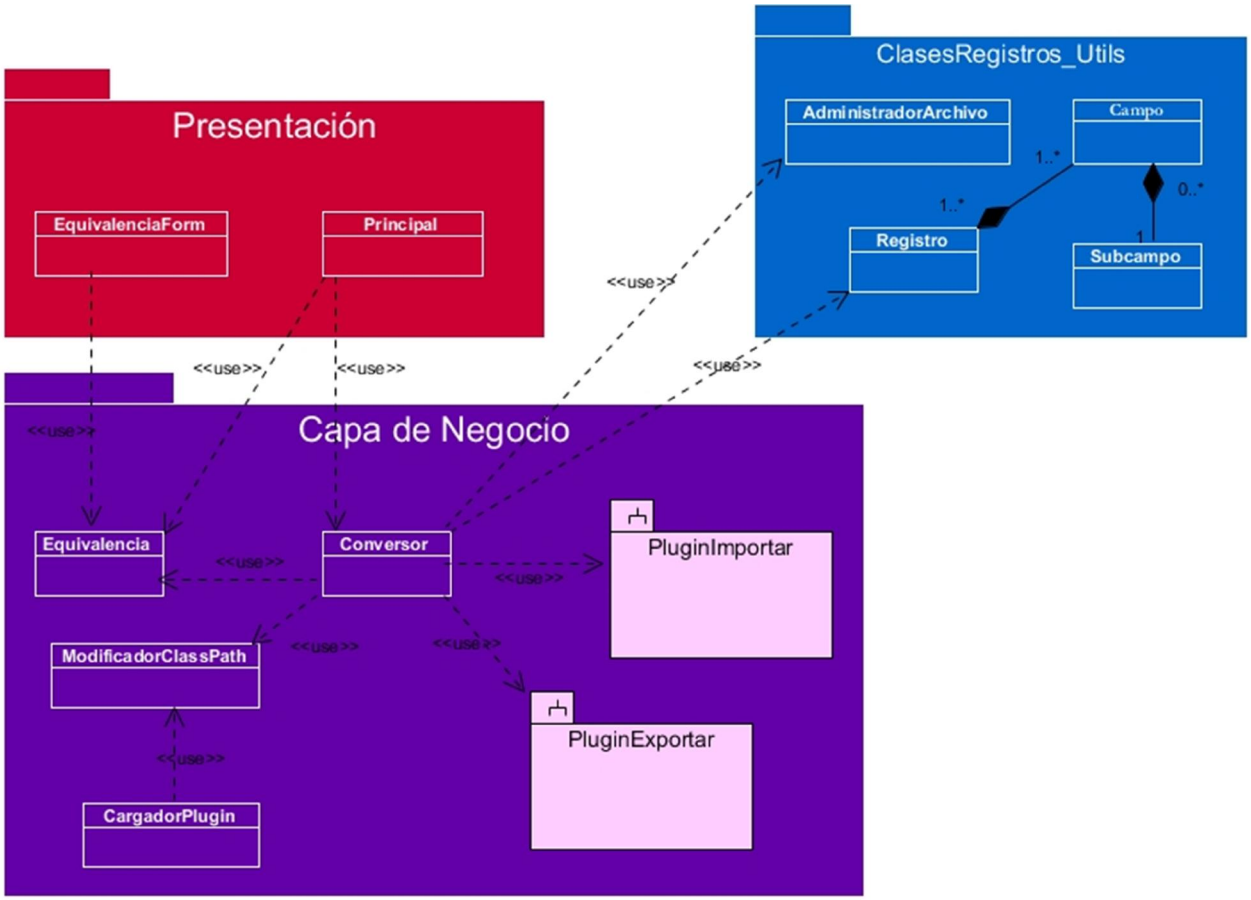


Figura 9 Diagrama de Clases del Diseño del Sistema

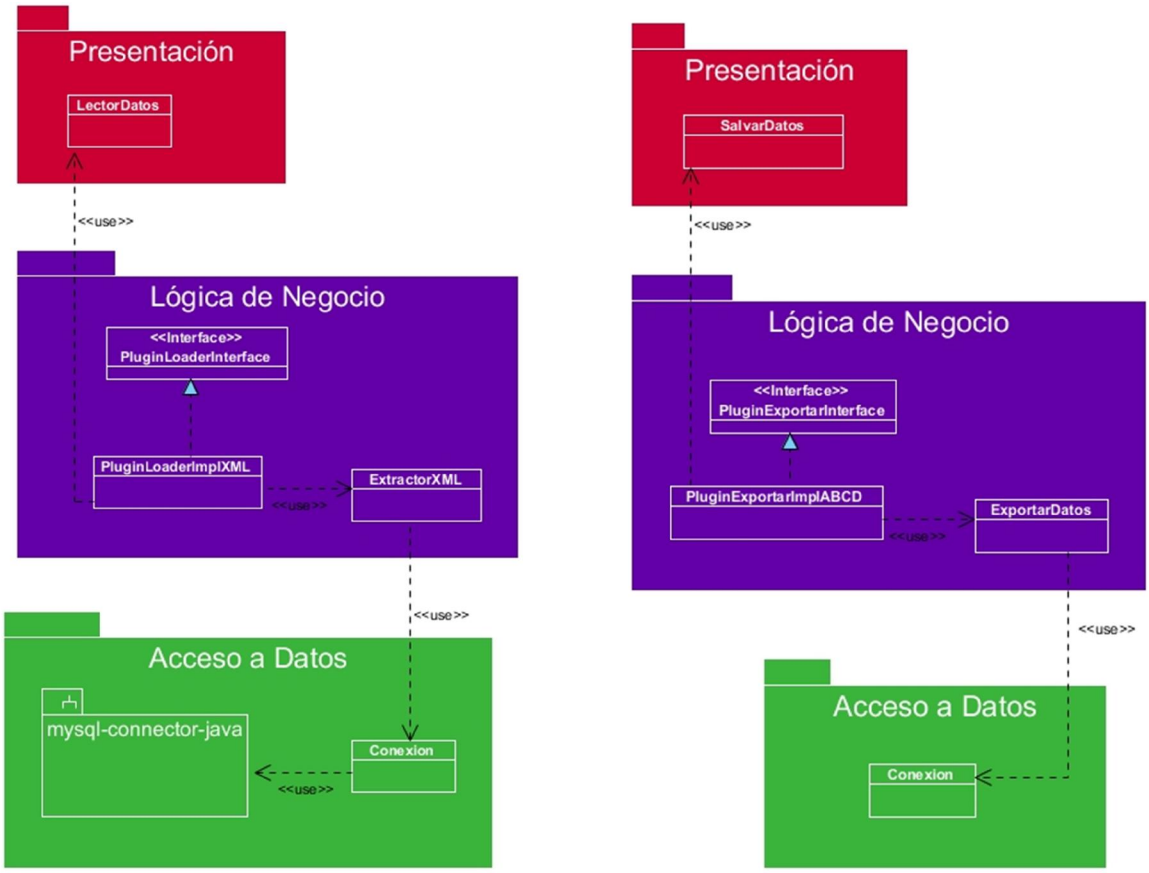


Figura 10 Diagrama de Clases del Diseño de los Plugin

A continuación se muestran las tarjetas CRC correspondientes a las clases más relevantes para las funcionalidades del Subsistema de extracción de registros bibliográficos para ver el resto remitirse al *Anexo 3*:

Clase PluginLoaderInterface	
Descripción: Esta clase brinda los métodos que serán implementados por los plugins.	
Responsabilidad	**Colaborador**
Mostrar interfaz principal	

Devolver el nombre a mostrar en el menú	

Tabla 8 Tarjeta CRC PluginLoaderInterface

Clase PluginLoaderImplXML	
Descripción: Esta clase se encarga de la implementación de la interfaz PluginLoaderInterface	
Responsabilidad	**Colaborador**
Mostrar la interfaz principal del plugin.	PluginLoaderInterface
Devolver el nombre definido para el plugin que se mostrará como opción en el menú principal de la aplicación.	

Tabla 9 Tarjeta CRC PluginLoaderImplXML

3.2. Desarrollo

Partiendo de los resultados hasta el momento, con el propósito de clarificar los requisitos restantes y completar el desarrollo del sistema basándose en la línea base de la arquitectura, se procede a la implementación, desarrollo o codificación del sistema. Esta fase se puede ver como un proceso de fabricación, en el que se pone énfasis a la gestión de los recursos y el control de las operaciones para optimizar los costes, la planificación y la calidad del producto final.

3.2.1. Estándares de codificación

Uno de los instrumentos que facilitan la tarea de asegurar la calidad del *software* es la adopción de estilos y estándares de codificación, es decir, técnicas de codificación sólidas apoyadas por buenas prácticas de programación. Con el uso de un estándar de codificación bien definido junto a revisiones del código se logra alcanzar un mejor rendimiento para determinada aplicación y caben muchas posibilidades de que un proyecto de *software* se convierta en una aplicación fácil de comprender y de mantener.

Hay que tener en cuenta que para el uso de un estándar de codificación se debe aplicar el mismo en todo momento, o sea, desde el principio hasta el final del proyecto de *software*. El uso de estos estándares

tiene innumerables ventajas entre ellas lograr un estilo de código homogéneo asegurando su legibilidad y proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.

Es debido a estos argumentos que la codificación de la solución propuesta debe cumplir con ciertos requisitos, algunos de los cuales son detallados seguidamente, el resto se encuentran en el *Anexo 4.*

- ✓ El código está estructurado en forma de bloques, esto favorece la mejor comprensión y lectura del mismo.
- ✓ Para evitar confusión de nombre y tipo se utilizan nombres que describen a los identificadores, en vez de nombres que describen el tipo de identificador.
- ✓ Se definen nombres de clases significativos, que expresan total o parcialmente su significado.
- ✓ Se emplean comentarios en determinadas declaraciones de clases y funciones más complejas.
- ✓ Para la capitalización de los nombres de las clases se utiliza el estilo Pascal y para los atributos el estilo Camello):
    - Pascal: la primera letra en el identificador y la primera letra de cada subsiguiente palabra concatenada se capitalizan. Puede utilizar los identificadores de Pascal case en caso de tres o más caracteres. Por ejemplo: *PluginLoaderInterface.*
    - Camello: La primera letra en el identificador está en minúscula y la primera letra de cada subsiguiente palabra concatenada en mayúscula. Por ejemplo: *fileManager.*
- ✓ Respecto a la sensibilidad a mayúsculas:
    - No se crean funciones con nombres de parámetros que se diferencian sólo en el uso de la mayúscula.
    - No se crean espacios de nombre con nombres de clases que se diferencien sólo en el uso de las mayúsculas.
    - No se crean clases con propiedades que se diferencien sólo en el uso de las mayúsculas.
    - No se crean clases con métodos que se diferencien sólo en el uso de las mayúsculas.

3.2.2. Tareas de Ingeniería

A lo largo del desarrollo de las iteraciones se realizará la implementación de las HU que componen cada una de estas. Se llevará a cabo una revisión del plan de iteraciones y se modificará este de ser necesario.

De acorde con el plan de desarrollo se descomponen las HU en tareas asignándolas posteriormente a un equipo (o una persona) responsable de su implementación. Estas tareas serán usadas por los programadores, pueden ser descritas utilizando un lenguaje técnico y no necesariamente deben ser entendibles para el cliente.

- ✓ **Número Tarea:** los números deben ser consecutivos.
- ✓ **Número Historia de Usuario:** número de la historia de usuario a la que pertenece la tarea.
- ✓ **Nombre Tarea:** nombre que identifica a la tarea.
- ✓ **Tipo de Tarea:** las tareas pueden ser de: Desarrollo, Corrección, Mejora, Otra (Especificar)
- ✓ **Puntos Estimados:** tiempo estimado en semanas que se le asignará a su desarrollo.
- ✓ **Fecha Inicio:** fecha en que inicia el desarrollo de la tarea.
- ✓ **Fecha Fin**: fecha en que finaliza el desarrollo de la tarea.
- ✓ **Programador Responsable:** nombre y apellidos del programador.
- ✓ **Descripción:** breve descripción de la tarea.

A continuación se describen algunas tareas de ingeniería, para ver el resto remítase al *Anexo 5:*

	Tarea de ingeniería
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Configurar plugin de entrada	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 4/2/2013	**Fecha Fin:** 15/2/2013
Programador Responsable: Andrés Sellés	
Descripción: El Gestor de migración introduce los parámetros para establecer la conexión a la base de datos donde se encuentran los registros.	

Tabla 10 Tarea de Ingeniería Configurar plugin de entrada

	Tarea de ingeniería
Número Tarea: 2	**Número Historia de Usuario:** 2
Nombre Tarea: Cargar registros bibliográficos	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1

Fecha Inicio: 18/2/2013	**Fecha Fin:** 23/2/2013
Programador Responsable: Andrés Sellés	
Descripción: El Conversor accede a la base de datos donde se encuentran los registros bibliográficos y los extrae.	

Tabla 11 Tarea de Ingeniería Transformar registros bibliográficos

3.2.3. Tratamiento de errores

Con el objetivo de garantizar la mayor confiabilidad e integridad en los datos que utiliza el sistema se adoptan las siguientes estrategias para el tratamiento de errores:

- ✓ Se validan los campos de los formularios de las interfaces pertenecientes los subsistemas de la aplicación
- ✓ La información que contiene los mensajes de errores detectados se muestra con un lenguaje claro, legible y sencillo de entender.

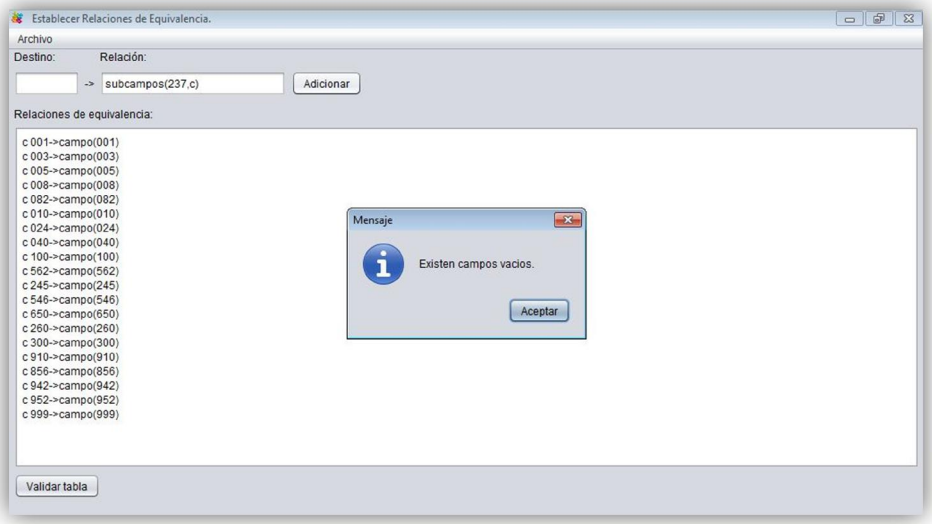


Figura 11 Ejemplo de tratamiento de errores

3.3. Pruebas

La prueba de *software* es un elemento crítico para la garantía de la calidad del mismo y representa una revisión final de las especificaciones del diseño y de la codificación. “El objetivo de la prueba es: diseñar pruebas que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio.” (36)

3.3.1. Pruebas Unitarias

Las pruebas unitarias son las encargadas de verificar el código y son diseñadas por los programadores. Cada uno de los desarrolladores tiene que ir probando constantemente lo que va obteniendo en el transcurso de la implementación de un sistema, para garantizar que las funcionalidades exigidas por el cliente estén siendo implementadas correctamente. Las pruebas unitarias fueron desarrolladas constantemente cada vez que se terminaba de implementar alguna funcionalidad probándola directamente en el entorno real.

Las pruebas unitarias proporcionan beneficios tales como:

- ✓ El programador puede realizar cambios de forma segura respaldada por efectivos casos de prueba.
- ✓ Permite saber si una determinada funcionalidad se puede agregar al sistema existente sin alterar el funcionamiento actual del mismo.
- ✓ Brindan al programador una inmediata retroalimentación de cómo está realizando su trabajo. (37)

Para realizar estas pruebas fue empleada la librería *JUnit*, reduciendo el tiempo y esfuerzo en esta fase, permitiendo que el desarrollador o probador se centre en la verificación de resultados correctos y no escribiendo código extenso para realizar sus pruebas.

JUnit proporciona clases de las cuales se pueden heredar para formar las nuevas clases que serán las que realicen las pruebas unitarias a cada una de las clases que conforman la aplicación. Estas pruebas están conformadas por una serie de datos y resultados, este último se compara con los datos que en realidad debería mostrar verificando si la aplicación cumple con lo solicitado por el cliente (38). A continuación se muestra un ejemplo de pruebas realizadas a la clase *FileManager:*

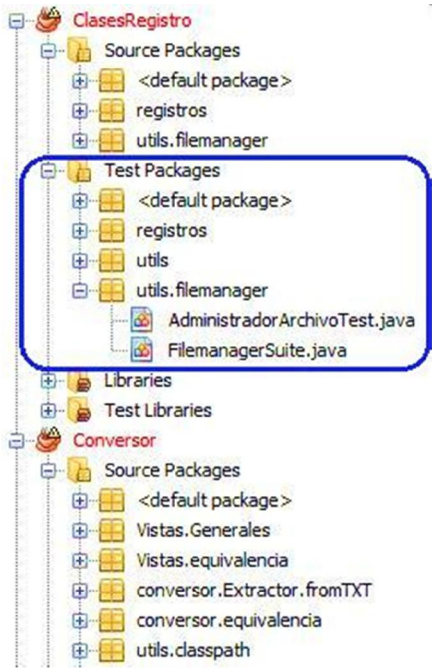


Figura 12 Ejemplo de Prueba Unitaria a la aplicación

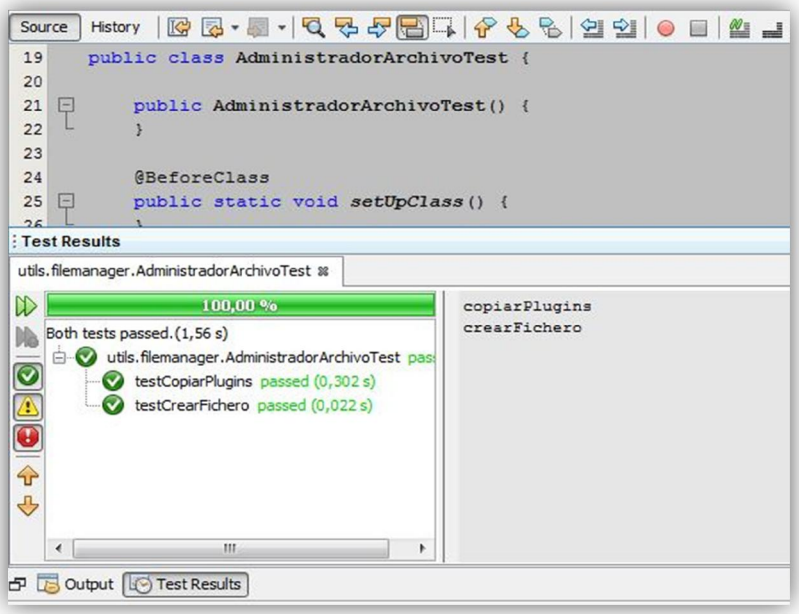


Figura 13 Prueba Unitaria a la clase *AdministradorArchivo*

3.3.2. Pruebas de Aceptación

Las pruebas funcionales son la mejor forma de probar la aplicación de extremo a extremo. Estas prueban todas las capas de la aplicación. En realidad, son muy similares a lo que se hace manualmente cada vez que se añade o modifica una acción y se prueban dichos cambios en la aplicación para comprobar que todo funciona bien al pulsar sobre los enlaces y botones y que todos los elementos se muestran correctamente en la página. En otras palabras, lo que se hace es probar un escenario correspondiente a la historia de usuario que se acaba de implementar en la aplicación.

Las pruebas de aceptación son consideradas como pruebas de caja negra (*Black box system tests*). Las pruebas de aceptación son tan importantes como las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente. Las pruebas de aceptación se elaboran a lo largo de la iteración, en paralelo con el desarrollo del sistema, y adaptándose a los cambios que el sistema sufra (37).
Dichas pruebas se diseñan en tablas, conocidas como Casos de Prueba las cuales están divididas en las siguientes secciones:

- ✓ **Clases Válidas**: Descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las entradas válidas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado.

- ✓ **Clases Inválidas**: Descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las posibles entradas inválidas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado y cómo responde el sistema.

- ✓ **Resultado Esperado**: Breve descripción del resultado que se espera ya sea para entradas válidas o entradas inválidas.

- ✓ **Resultado de la Prueba**: Breve descripción del resultado que se obtiene.
- ✓ **Observaciones**: algún señalamiento o advertencia que sea necesario hacerle a la sección que se está probando.

- ✓ **Evaluación de la prueba:** Acorde al resultado de la prueba realizada se emite una evaluación sobre la misma:

- Bien: Cuando el resultado de la prueba es exactamente el esperado por el usuario.
- Parcialmente bien: Cuando el resultado no es completamente el esperado por el cliente o usuario de la aplicación y muestra resultados erróneos o fuera de contexto.
- Mal: Cuando el resultado de la prueba realizada genera un error de codificación en la aplicación o muestra como resultado elementos no deseados o fuera de contexto, trayendo como consecuencia que la funcionalidad requerida por el cliente no tenga resultado, lo que invalida también la HU.

Las pruebas de aceptación correspondiente a las funcionalidades del Subsistema de extracción de registros bibliográficos pueden encontrarse en el *Anexo 6.*

Luego de realizarse las pruebas de aceptación se obtienen un total de 23 no conformidades donde 16 fueron resueltas, 7 no procedían y no quedó ninguna pendiente. A continuación se ilustran los resultados de estas pruebas en la gráfica de la *figura :*

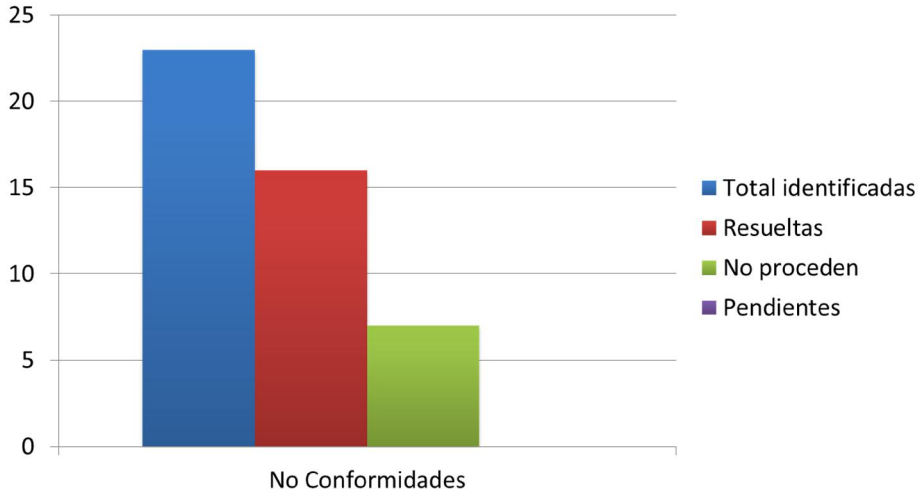


Figura 14 Resultados de las Pruebas de Aceptación

Conclusiones parciales

En este capítulo se realizó el diseño del patrón arquitectónico definido para el desarrollo de la aplicación. También se plantearon los problemas de diseño e implementación principales para darle solución mediante los patrones de diseño. Se describieron e identificaron las tareas de la ingeniería correspondientes a las HU descritas en el anterior capítulo. Por último se llevaron a cabo las pruebas del sistema logrando con estas la satisfacción del cliente con el software realizado.

CONCLUSIONES GENERALES

Al concluir investigación que trajo consigo el desarrollo de la aplicación informática que permitirá la migración dinámica entre formatos bibliográficos entre el Sistema Integrado de Gestión Bibliotecaria y el sistema ABCD, se pueden arribar a las siguientes conclusiones:

- ✓ El estudio del arte estuvo enfocado hacia herramientas que realizan la conversión de ficheros bibliográficos y herramientas de migración de datos, arrojando como resultado que ninguna brinda una solución para la problemática planteada y demostrando la necesidad de elaborar una propuesta de solución ajustada a las especificaciones del cliente.

- ✓ Las actividades de análisis y diseño orientadas a la modelación de la aplicación informática para la migración dinámica entre formatos bibliográficos permitió analizar los procesos que componen dicha herramienta y diseñar la arquitectura definida para la misma.

- ✓ Se realizó la implementación del software para la migración dinámica entre formatos bibliográficos siguiendo las pautas de la arquitectura descrita, logrando cumplir con los objetivos trazados por el equipo de trabajo y con las especificaciones del cliente.

- ✓ Con la puesta práctica de la aplicación se logrará la migración de los registros bibliográficos que se encuentran en la base de datos del SIGB KOHA hacia la base de datos CDS/ISIS de ABCD.

- ✓ Quedó demostrado el correcto funcionamiento de la aplicación a través de las pruebas realizadas durante cada ciclo de desarrollo y al producto final.

RECOMENDACIONES

A partir de los resultados o beneficios que proporciona este trabajo de diploma, se proponen las siguientes recomendaciones:

- ✓ Que se despliegue la primera versión de la aplicación en el proyecto ABCD.
- ✓ Investigar otras posibles funcionalidades y que se realice la implementación de las mismas.
- ✓ Implementar nuevos plugins que permitan realizar otras migraciones como es el caso de los registros existentes en la Biblioteca Nacional.

REFERENCIAS BIBLIOGRÁFICAS

1. **Casson, Lionel.** *Libraries in the ancient world.* New Haven and London : Yale University Press, 2001.

2. **GARCÍA CAMARERO, E. and GARCÍA MELERO, L.A.** *Automatización de bibliotecas.* Madrid : Arco/Libros, 1999.

3. **COMMUNITY, KOHA.** [Online] [Cited: Enero 11, 2013.] http://es.koha-community.org/.

4. NEWgen Coding Library. [Online] Copyright NewGenLib.com. [Cited: Enero 27, 2013.] http://www.newgenlib.com/.

5. OpenBiblio a library system that´s free. [Online] [Cited: Enero 27, 2013.] http://obiblio.sourceforge.net/index.php/Main/OpenBiblio.

6. *Formatos Bibliográficos: El IBERMARC.* **Cuesta Escudero, María Jesús.** 1976.

7. Edutiva. [Online] [Cited: Enero 7, 2013.] http://www.edutiva.com/modulo-de-biblioteca.htm.

8. **Rodríguez Castilla, Liuris.** *El sistema Integrado de Gestión Bibliotecaria: su implementación en la Biblioteca de la Universidad de las Ciencias Informáticas (BiUCI).* La Habana : Facultad de Comunicación, 2003. Tesis.

9. **Principles, The MARC 21 Formats: Background and.** MARC STANDARDS. [Online] [Cited: Diciembre 6, 2012.] http://www.loc.gov/marc/96principl.html.

10. **Lapuente, María Jesús Lamarca.** *Hipertexto, el nuevo concepto de documento en la cultura de la imagen.* Facultad de Ciencias de la Información. Dpto. de Biblioteconomía y Documentación, Universidad Complutense de Madrid. Tesis Doctoral.

11. *Un Generador Automático de Planes de Migración de Datos. I+D Computación.* **CARSÍ, J. A., RAMOS, I. and SILVA, J.** 1, Valencia : Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Julio 2002, Vol. 1. Disponible en: http://issi.dsic.upv.es/publications/archives/f-1050406290873/No1Art02.pdf.

12. Talend. [Online] Diciembre 10, 2012. http://es.talend.com/solutions-data-integration/data-migration.php..

13. **CROSS, DAVID. I.** *Data Munging with Per.* s.l. : Manning Publications Co, 2001.

14. Biblioteca Nacional De España(BNE). [Online] [Cited: Diciembre 7, 2012.] http://www.bne.es.

15. UNESCO. [Online] [Cited: Diciembre 7, 2012.] http://www.unesco.org/isis/files/winisis/windows/utilities/.

16. Pentaho Community. [Online] [Cited: Marzo 13, 2013.] http://wiki.pentaho.com/display/EAIes/.03+Conexiones+a+Bases+de+Datos.

17. **Iglesias, Rodolfo.** Oracle vs Oracle. [Online] Septiembre 2008. http://www.oracle.com/technetwork/es/documentation/317445-esa.pdf.

18. *Aplicando el método de Boehm y Turner.* **Boeras Velázquez, Mairelys, et al., et al.** 6, Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2012, Vol. 5. Disponoble en: http://publicaciones.uci.cu/index.php/SC| seriecientifica@uci.cu. ISSN: | RNPS:.

19. *Procesos de desarrollo: RUP, XP, FDD.* **Molpeces, Alberto.** México : s.n., Diciembre 15, 2003. Disponible en: [www.willydev.net/descargas/Articulos/General/cualxpfddrup.PDF].

20. **Schmuller, Josep.** APRENDIENDO UML EN 24 HORAS. Mexico: PEARSON EDUCACION, 1999 .

21. **Larman, Craig.** *UML y Patrones.* Naucalpan de Juhez : Prentice Hall, Prentice Hall.

22. Visual Paradigm. [Online] [Cited: Enero 8, 2013.] http://www.visual-paradigm.com/product/vpuml/.

23. **Albahari, Ben.** *A Comparative Overview of C#.* s.l. : Genamics, 2000.

24. **Grand, Mark.** *Java Language Reference.* 1st Edition. O'Reilly : s.n., 1997.

25. JAVA. [Online] [Cited: Enero 12, 2013.] http://www.java.com.

26. Oracle Corporation. [Online] [Cited: Diciembre 7, 2012.] http://neatbeans.org/.

27. Synergix. Wordpress. [Online] [Cited: Febrero 9, 2013.] http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.

28. **Beck, Kent.** *Extreme Programming Explained.* s.l. : Addison Wesley The XP Series, 2000.

29. **Camacho, Erika, Cardeso, Fabio and Nuñez, Gabril.** *Arquitecturas de Software.* 2004.

30. **Acebal, César F. y Cueva Lovelle, Juan M.** *Extreme Programming (XP): un nuevo método de desarrollo de software.* Oviedo: Novatica, 2002.

31. **Meyer, Bertrand .** *Construcción de software orientado a objetos.* Segunda edición. s.l. : Prentice Hall.

32. *Architectural Blueprints--The 4+1 View Model of Software Architecture.* **Kruchten, Philippe.** New York : IEEE Software, 1995.

33. **Christopher Alexander.** *A Pattern Language: Towns, Buildings, Construction.* California : Oxford University Press, 1977. ISBN: 0-19-501919-9.

34. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : PRENTICE HALL, 1999. ISBN: 970-17-0261-1.

35. **Escribano, G. F.** *eXtreme Programming / Programación Extrema.* 2002.

36. **Crispin, Lisa and House, Tip.** Testing Extreme Programming. s.l. : Addison-Wesley, 2002.

37. **Pérez, Isaías Carrillo.** METODOLOGIA DE DESARROLLO DE SOFTWARE. 2005.

38. **Ramos, Rafael Aldo.** compujuy. [Online] 2013. [Cited: mayo 13, 2013.] http://www.compujuy.com.ar/.

39. *El z39.50 en el ambiente de transferencia y recuperacion de informacion.* **Arango, Martha Elena.** s.l. : Pontificia Universidad Javeriana.

BIBLIOGRAFÍA CONSULTADA

1. Edutiva. [Online] http://www.edutiva.com.

2. Talend. [Online] http://es.talend.com/solutions-data-integration/data-migration.php.

3. **CROSS, DAVID. I.** *Data Munging with Per.* s.l. : Manning Publications Co, 2001.

4. NetBean. [Online] http://www.netbean.org.

5. Eclipseos. *Eclipseos.* [Online] http://www.eclipseos.es/soluciones/gis/.

6. **Reitz, J. M.** [Online] 2004. http://www.abc-clio.com/ODLIS/odlis_m.aspx.

7. Unesco. [Online] .http://portal.unesco.org/ci/en/ev.php-URL_ID=5330\&URL_DO=DO_TOPIC\&URL_SECTION=201.html.

8. **Coad, Peter, Lefevre, Eric y De Luca, Jeff.** *JAVA Modeling COLOR with UML*. s.l.: Prentice Hall PTR, 1999. pág. 248. ISBN 13: 9780130115102.