Universidad de las Ciencias Informáticas
Facultad 6

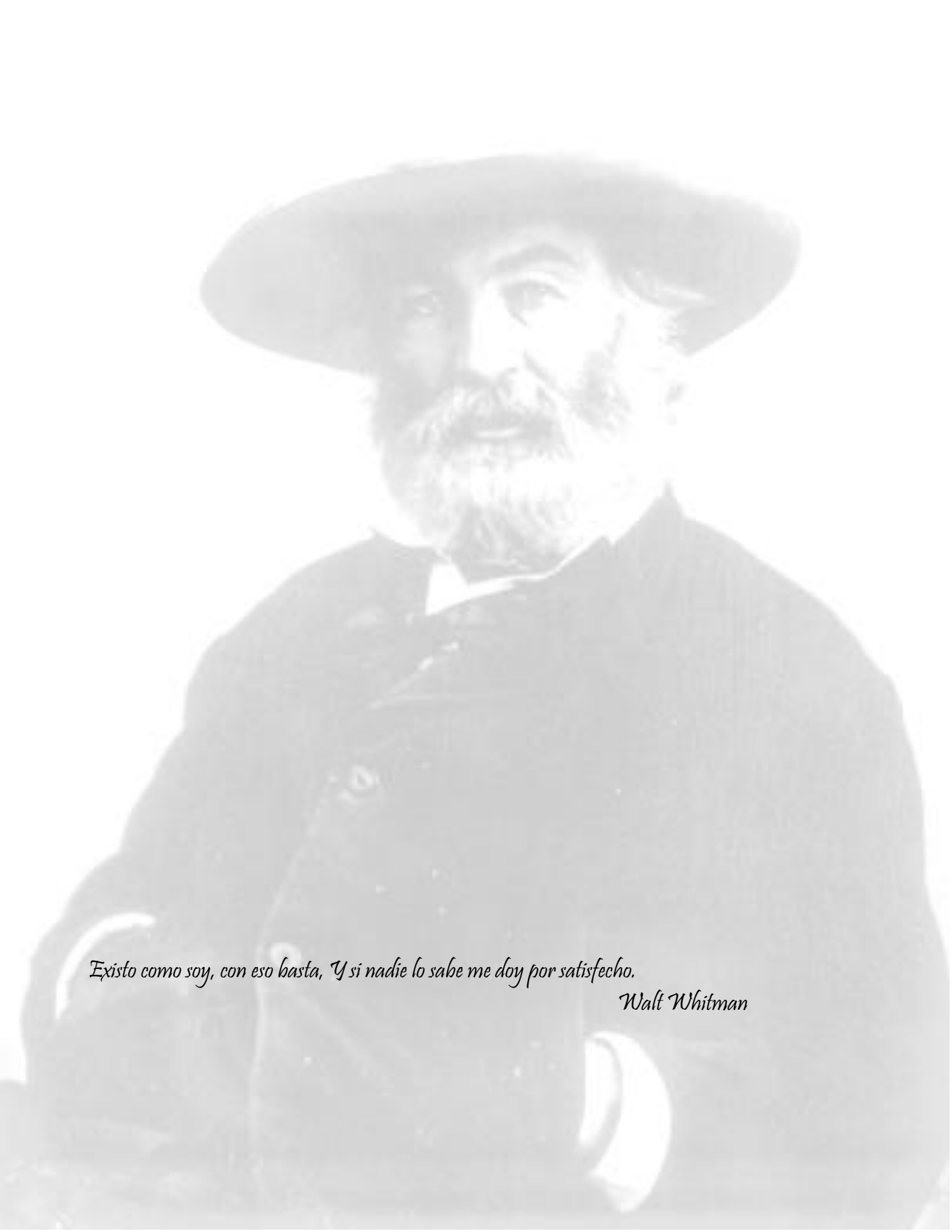
Título: Extensión para la gestión del Mapa Lógico de Datos en la herramienta de modelado “Visual Paradigm for UML”.

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

Autores: Eudel Campuzano Fuentes
Liena Calzado Robaina

Tutores: Ing. Yanisbel González Hernández
Ing. Ramón Ernesto Stevenson Borrell

La Habana, 2013
“Año 55 de la Revolución.”



Existo como soy, con eso basta, Y si nadie lo sabe me doy por satisfecho.

Walt Whitman

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Eudel Campuzano Fuentes

_____

Firma del autor

Liena Calzado Robaina

_____

Firma del autor

Ing. Yanisbel González Hernández

_____

Firma del tutor

Ing. Ramón Ernesto Stevenson Borrell

_____

Firma del tutor

DATOS DE CONTACTOS

Tutores: Ing. Yanisbel González Hernández
Categoría Docente: Instructor
Ingeniera Informática, Universidad de Ciencias Informáticas, 2006
Correo Electrónico: yglezh@uci.cu

Ing. Ramón Ernesto Stevenson Borrell
Ingeniero Informático, Universidad de Ciencias Informáticas
Correo Electrónico: restevenson@uci.cu

Autores: Eudel Campuzano Fuentes
Universidad de Ciencias Informáticas
La Habana, Cuba
Correo Electrónico: ecampuzano@estudiantes.uci.cu

Liena Calzado Robaina
Universidad de Ciencias Informáticas
La Habana, Cuba
Correo Electrónico: lcalzado@estudiantes.uci.cu

RESUMEN

La integración de datos es el proceso de unificación de los datos provenientes de múltiples fuentes. Este proceso se puede efectuar de diferentes formas, una de ellas es la extracción, transformación y carga de la información (ETL), que permite a las organizaciones mover información desde diversas fuentes, reformatearla, limpiarla y cargarla en otras bases de datos. Un elemento esencial para el diseño del subsistema de integración en el departamento Almacenes de datos, es la confección del artefacto Mapa Lógico de Datos, cuyo propósito principal es documentar de manera inequívoca la localización de los datos en los sistemas origen (fuentes de datos), posibilitando además, gestionar adecuadamente el análisis de las fuentes y establecer el flujo de la información hasta el destino (almacén de datos). En el presente trabajo se realiza la implementación de una extensión para la gestión del Mapa Lógico de Datos en la herramienta CASE de modelado “Visual Paradigm for UML”, con el objetivo de facilitar la elaboración de este artefacto.

Palabras clave: CASE, ETL, Integración de datos, Mapa Lógico de Datos.

TABLA DE CONTENIDOS

Tabla de contenidos

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Al constituir una parte esencial de la actividad humana en cualquiera de sus manifestaciones, la información acumulada se revela como un activo útil y de obligado interés para la sociedad moderna. La capacidad de procesar, asimilar, innovar, compartir y generar nuevos conocimientos a través de su estudio, constituye uno de los grandes retos de las actuales Tecnologías de la Informática y las Comunicaciones (TIC[1]).

Los avances logrados en los distintos campos de las TIC posibilitan un incremento sustancial en el desempeño humano en todo su universo de acción, incorporando a su vez, nuevas problemáticas asociadas al uso intensivo de dichas tecnologías. Las tecnologías de la información han sido conceptualizadas como la integración y convergencia de las telecomunicaciones, la informática y el audiovisual (1). La Informática es una Ciencia o Disciplina con principios, métodos, técnicas y herramientas propias que facilita captar y estudiar los fenómenos relacionados con el tratamiento sistemático de la información (2). La misma tiene un papel importante y rector en el grupo de ciencias que conforman la base para estas nuevas tecnologías.

El desarrollo exponencial de estas ciencias en las últimas décadas ha marcado incluso un punto de comparación incuestionable entre los que tienen acceso a ellas y los que no, para determinar su capacidad de gestión en numerosas áreas. Cuba, como nación, ha realizado incontables esfuerzos para mantener esta brecha controlada dentro de los límites que le permite el contexto socio-económico en que se encuentra. Con este objetivo se constituye la Universidad de las Ciencias Informáticas (UCI), cuya meta fundamental es graduar un personal altamente calificado orientado a informatizar la sociedad cubana.

La UCI implementa un nuevo paradigma para este tipo de industria dentro del contexto cubano, en este se vincula el concepto de estudio-trabajo de un modelo educacional con un enfoque de producción orientado a centros de desarrollo. En este ámbito se aprovechan los estrechos vínculos generados entre las facultades y áreas temáticas especializadas denominadas centros de producción de software. Dichos centros enlazan la producción, la investigación y la formación de estudiantes como futuros profesionales altamente calificados a sus actividades de desarrollo de software. Entre estos centros se encuentra el

[1]Toda forma de tecnología usada para crear, almacenar, procesar e intercambiar información.

Centro de Tecnologías de Gestión de datos (DATEC), el cual tiene como misión proveer soluciones integrales, así como, consultorías relacionadas con tecnologías de bases de datos y el análisis de información.

DATEC está compuesto por cuatro líneas de productos de software: PostgreSQL, Bioinformática, Soluciones integrales y Almacenes de datos. En cada una de las líneas se hace uso de un conjunto de herramientas informáticas con el objetivo de facilitar el proceso de desarrollo de software. Una de estas herramientas es "Visual Paradigm for UML" (VP), la cual ofrece entre sus funcionalidades la representación de modelos de software mediante modelos UML[2].

El departamento almacenes de datos (DAD) utiliza esta herramienta para todo el ciclo de desarrollo de sus soluciones. Dentro de las políticas de trabajo establecidas está el uso de herramientas libres para el desarrollo de sus productos en concordancia con las políticas y objetivos orientados por la dirección del país, con el fin de establecer en un tiempo prudencial una soberanía tecnológica en nuestra infraestructura de información. Estas soluciones de desarrollo de almacenes de datos se dividen en tres subsistemas fundamentales: subsistema de integración de datos, subsistema de almacenamiento y subsistema de visualización de información.

La presente investigación se centra en una problemática específica del grupo ETL perteneciente al DAD. Un elemento esencial para el diseño del subsistema de integración de datos es la confección del Mapa Lógico de Datos (MLD). Este artefacto describe el flujo de los datos desde el origen (fuentes de datos) hasta el destino (almacén de datos). En la actualidad para desarrollar este artefacto se utiliza un fichero o archivo en formato Excel diseñado por el departamento, en el cual deben introducirse de forma manual el nombre de todas las tablas y atributos de las fuentes de datos y del almacén de datos para después relacionarlos. Esto hace que la realización de este artefacto resulte engorrosa, requiera de mucho tiempo para su desarrollo y esté propenso a la introducción de errores humanos.

La bibliografía especializada (3) reconoce el subsistema de integración de datos como el que más tiempo y esfuerzo requiere en la construcción de almacenes de datos. Los retrasos o problemas incorporados en este punto afectan considerablemente los tiempos de desarrollo de la solución en general, por lo que mitigar este riesgo constituye un elemento de crucial interés en la gestión de este tipo de proyectos.

[2]Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language)

Por lo anteriormente planteado se identifica el siguiente **problema de la investigación:** ¿Cómo automatizar el proceso de mapeo lógico de datos utilizado por el Departamento Almacenes de datos del centro DATEC?

La investigación tiene como **objeto de estudio:** procesos de integración de datos, enmarcado en el **campo de acción:** desarrollo de extensiones para la herramienta “Visual Paradigm for UML” para la integración de datos.

Con el propósito de solucionar el problema anteriormente planteado, la presente investigación tiene como **objetivo general**: desarrollar una extensión de la herramienta “Visual Paradigm for UML” para la elaboración del Mapa Lógico de Datos, el cual se desglosa en los siguientes **objetivos específicos:**

1. Realizar el estudio de los procesos de integración de datos, describir las estructuras de datos que debe contemplar la extensión, además del entorno de desarrollo de extensiones para “Visual Paradigm for UML, la metodología y herramientas para el desarrollo de la extensión.
2. Realizar el análisis y diseño de la extensión para la elaboración de la solución.
3. Realizar la implementación y prueba de la extensión para la elaboración del Mapa Lógico de Datos.

Los objetivos específicos anteriormente detallados serán cumplidos a través de las **tareas de la investigación** enunciadas a continuación:

1. Caracterización de los procesos de integración de datos.
2. Descripción de los principales formatos que se van a incluir en el desarrollo de la extensión.
3. Revisión bibliográfica de la documentación técnica de “Visual Paradigm for UML” referida a la extensión de la herramienta.
4. Análisis y selección de las metodologías, herramientas y tecnologías a utilizar en el desarrollo de la extensión.
5. Análisis de la extensión para la elaboración de la solución.
6. Diseño de la solución.
7. Implementación de la extensión para la elaboración del Mapa Lógico de Datos.
8. Realización de pruebas de aceptación por parte del Departamento de Almacenes de Datos.

El documento está estructurado en tres capítulos que se describen a continuación:

Capítulo1: Fundamentación teórica de la extensión para la elaboración del Mapa Lógico de Datos

En este capítulo se abordarán los principales conceptos que serán de utilidad para el dominio de la presente investigación. Se describen las estructuras de datos que debe contemplar la extensión y se seleccionarán las herramientas, lenguaje de programación y metodología para el desarrollo de la investigación.

Capítulo2: Análisis y diseño de la extensión para la elaboración del Mapa Lógico de Datos

En este capítulo se describe la propuesta de solución para la situación problemática anteriormente planteada. Se definirán las principales características que poseerá la aplicación informática a desarrollar, comenzando con los elementos arquitectónicos a tener en cuenta para la implementación de extensiones en VP, el estudio de la planificación, la definición de los requisitos funcionales, no funcionales y el diseño.

Capítulo3: Implementación y prueba de la extensión para la elaboración del Mapa Lógico de Datos

En este capítulo se realizará la descripción de las principales funcionalidades codificadas para la extensión, las cuales estarán guiadas por un conjunto de estándares de codificación, y se validará la calidad del software mediante la realización de pruebas al mismo.

CAPÍTULO 1. FUNDAMENTO TEÓRICO DE LA EXTENSIÓN PARA LA ELABORACIÓN DEL MAPA LÓGICO DE DATOS

En el presente capítulo se describen los aspectos teóricos relacionados con el desarrollo de extensiones o "plugins" incorporados a la Suite Visual Paradigm. Se caracterizan y describen los conceptos de integración de datos y Mapa Lógico de Datos. Se presentan las metodologías y herramientas existentes para el desarrollo de este tipo de soluciones, y se exponen conceptos importantes que permiten una comprensión significativa de la investigación y justifican la solución escogida.

1.1 Procesos de integración de datos

El proceso de integración de datos está sustentado por la necesidad de reunir los datos de diferentes fuentes de información. A este concepto se le han señalado varias definiciones, como la adjudicada a Ralph Kimball, un reconocido especialista en la materia donde plantea: *"la Integración de datos es el proceso de unificación de los datos provenientes de múltiples fuentes…Esto provoca que la información proveniente de los sistemas externos sea inconsistente y de baja calidad, convirtiendo este proceso en un trabajo costoso y complejo"*. (4)

La integración de datos en términos tecnológicos se puede enfocar de varias formas, dependiendo de la idea de "Integración" que se tenga, los objetivos que se persigan y las condiciones existentes en el contexto de trabajo. Fundamentalmente existen cuatro tipos:

1. **Replicación de Datos:** es mucho más que la simple copia de datos entre varios dispositivos. Ha sido utilizada, tradicionalmente, como el mecanismo básico para incrementar la disponibilidad y rendimiento de una Base de Datos Dinámica. (5)
2. **Integración de Aplicaciones Empresariales (EAI):** es el proceso de integrar múltiples aplicaciones desarrolladas independientemente, que utilizan tecnología incompatible y que son gestionadas de forma independiente, permitiendo que se comuniquen e intercambien transacciones de negocio, mensajes, y datos entre sí. Uno de los principales objetivos de EAI es proporcionar acceso transparente a la amplia gama de aplicaciones que existen en una organización. (6)

3. **Integración de Información Empresarial (EII):** permite la integración de datos procedentes de múltiples sistemas en una representación única, coherente y orientada hacia la visualización y manipulación de los datos. (7)
4. **Extracción, Transformación y Carga de Datos (ETL):** estos procesos se combinan para extraer datos de bases de datos fuentes, archivos u otro sistema, y colocarlas en bases de datos destino. Es la tecnología enfocada a la Integración de datos, tanto por lote como a tiempo real hacia Almacenes de Datos. (3)

De estos enfoques el más conocido y aplicado en el área de transformación de los datos es ETL. Este término constituye una amplia categorización de las actividades de integración de datos, ya que es el enlace entre las fuentes y el almacén de datos. Concretamente se distinguen: (8)

- **Extracción:** Normalmente los almacenes de datos consolidan diferentes sistemas de fuentes de datos. Cada sistema separado puede usar una organización diferente de los datos o formatos distintos. Los formatos de las fuentes normalmente se encuentran en bases de datos relacionales o ficheros planos, pero pueden incluir bases de datos no relacionales u otras estructuras diferentes. La fase de extracción convierte los datos de los diferentes sistemas, a un formato preparado para iniciar el proceso de transformación. Al mecanismo para especificar las correspondencias o mapeos entre el esquema fuente y un esquema intermedio para cargar la información en el almacén se denomina *mapping* o mapeo. Estos mapeos son cálculos y funciones y se considera parte del código o metadatos del almacén.
- **Transformación:** La fase de transformación aplica una serie de reglas de negocio o funciones sobre los datos extraídos para convertirlos en datos que puedan ser cargados. Algunas fuentes de datos requerirán alguna pequeña manipulación de los datos. Algunas de las transformaciones más sencillas pueden ser:
  - ✓ Seleccionar solo ciertas columnas para su carga (o si lo prefiere, que las columnas con valores nulos no se carguen)
  - ✓ Traducir códigos (Ej. Si la fuente almacena una "H" para Hombre y "M" para Mujer pero el destino tiene que guardar "1" para Hombre y "2" para Mujer)
  - ✓ Unir datos de múltiples fuentes (ej. búsquedas y fusión)
  - ✓ Sumar múltiples filas de datos (ej. ventas totales de cada región)

- ✓ Generación de campos clave en el destino

- **Carga:** La fase de carga es el momento en el cual los datos de la fase anterior son cargados en el destino. Dependiendo de los requerimientos de la organización, este proceso puede abarcar una amplia variedad de procesos diferentes. Algunos almacenes sobrescriben información antigua con nuevos datos. Los sistemas más complejos pueden mantener un historial de los registros de manera que se pueda hacer una auditoría de los mismos y disponer de un rastro de toda la historia de un dato, lo que se denomina seguimiento de cambios.

En los proyectos de desarrollo de este tipo de soluciones, la documentación asociada contiene un alto valor para garantizar la correcta gestión y éxito del proceso de integración de datos. Dependiendo de la metodología seleccionada existen muchas variantes sobre cómo realizar este registro documental de forma tal que sea útil para los especialistas que intervienen, y se traduzcan en un artefacto de referencia para futuros desarrollos. La reusabilidad y simplificación en la generación de los registros juega un papel fundamental en la mejora de este proceso.

Un elemento clave para el desarrollo del proceso de ETL según la metodología utilizada por el departamento Almacenes de datos de DATEC es la confección del Mapa Lógico de Datos. Este artefacto en conjunto con las especiaciones funcionales proporciona a los desarrolladores una visión clara de lo que se espera del proceso de integración, además de los elementos necesarios para la implementación del mismo.

1.2 Mapa Lógico de Datos

El Mapa Lógico de Datos constituye la base para el desarrollo del proceso de escenificación de los datos. Su propósito principal es documentar de manera inequívoca la localización de los datos en los sistemas fuentes, posibilitando además, gestionar adecuadamente el análisis de las fuentes y su correspondiente documentación. El mapa completo incluye elementos de datos que se encuentran en la fuente, pero no requieren ser incluidos en el almacén de datos, esta información adicional puede ser útil en la comprensión de las potencialidades de información subyacente existente en los sistemas fuentes y su inclusión en futuros requerimientos. El MLD se presenta generalmente en formato de una tabla u hoja de cálculo Excel.

En el DAD incluye los siguientes componentes:

- **Nombre de la Base de Datos Fuente:** se especifica el nombre de la base de datos fuente.
- **Nombre del Esquema Fuente:** se especifica el esquema al que pertenece la tabla que tiene la información en la base de datos fuente.
- **Nombre de la Tabla Fuente:** se especifica el nombre de la tabla que tiene la información en la base de datos fuente.
- **Nombre de la Columna Fuente:** se especifica el nombre de la columna en la tabla fuente.
- **Tipo de Dato:** Se especifica el tipo de dato de la columna.
- **Nombre de la Tabla Destino:** en esta columna se especifica el nombre de la tabla destino.
- **Nombre de la Columna Destino:** se especifica el nombre de la columna destino en la tabla destino.
- **Tipo de Dato:** se especifica el tipo de dato de la columna destino en la tabla destino.
- **Tipo de Tabla:** se especifica el tipo de tabla destino (dimensional o hecho).
- **SCD (Dimensiones Lentamente Cambiantes):** se especifica el tipo de SCD para las tablas dimensionales: definidos desde el tipo 0 hasta el 6, sin incluir el 5.

Para evidenciar de manera más clara la composición estructural del Mapa Lógico de Datos, a continuación se muestra la **Figura 1**, en la cual se observa la forma en la que se agrupan dentro de la fuente y el destino los componentes antes descritos:

Figura 1 Componentes del Mapa Lógico de Datos

Los componentes individuales en el mapeo de datos lógicos parecen ser simples y directos. Sin embargo, cuando se estudia con más detenimiento, el documento revela muchos requisitos importantes para el equipo de ETL que de otro modo podrían haber pasado por alto. Uno de los propósitos de este artefacto es proporcionar al desarrollador un anteproyecto claro de lo que se espera del proceso de integración, para ello es necesario contar con un registro detallado de los tipos de orígenes o fuentes de datos.

1.3 Orígenes o fuentes de datos

Los orígenes de datos son los sistemas encargados de recoger la información de las transacciones generadas en la organización. Estos sistemas se conocen habitualmente como sistemas operacionales. Deben ser sistemas fiables y consistentes, aunque entre ellos haya marcadas diferencias en los formatos y las estructuras de los datos. Estos sistemas quedan fuera del almacén por lo que no se tiene el control sobre el contenido de sus datos. (8)

Estos constituyen estructuras de datos que contienen información necesaria para el proceso de integración. A continuación se describen las fuentes definidas en el alcance de la tesis según los intereses del departamento Almacenes de datos:

- **Sistema Gestor de Base de Datos**: los Sistemas Gestores de Bases de Datos (SGBD[3]) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. El propósito general de los sistemas de gestión de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convierten en información relevante (9).

  La universidad se encuentra inmersa en un proceso de migración, vinculada principalmente a las nuevas tendencias tecnológicas, que consiste en el uso de políticas de software libre. Dentro de las herramientas libres con las que trabaja la universidad se tiene dominio del conocimiento asociado a la implementación del proceso de conexión de algunas de estas, por lo que se seleccionaron PostgreSQLy MySQL como sistemas gestores de bases de datos. En la aplicación se utilizarán como fuentes de datos los ficheros sql generados por los SGBD escogidos, debido a que fueron concebidos así en el análisis de selección de los orígenes a cargar.

- **XML:** Siglas en inglés de Extensible Markup Language ('lenguaje de marcas extensible'). Permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información. Juega un papel importante en la actualidad ya que es utilizada para ofrecer compatibilidad entre sistemas heterogéneos para compartir información de

[3]En numerosas literaturas se conocen por las siglas DBMS, del inglés Data Base Management System.

una manera segura, fiable y fácil. La fuente XML puede ser un archivo al que se puede acceder desde el sistema de archivos local o a través de una red. (10)

- **Hoja de cálculo:** Una hoja de cálculo es un programa o aplicación, que permite manipular datos numéricos y alfanuméricos dispuestos en forma de tablas compuestas por celdas (las cuales se suelen organizar en una matriz bidimensional de filas y columnas). La celda es la unidad básica de información en la hoja de cálculo, donde se insertan los valores y las fórmulas que realizan los cálculos. Habitualmente es posible realizar cálculos complejos con fórmulas, funciones y dibujar distintos tipos de gráficas. (11)

Luego de describir las estructuras de datos que se incluirán en la extensión, es importante validar la fundamentación teórica de la selección de especificaciones técnicas del proyecto. Para lo cual es necesario realizar un estudio del entorno de desarrollo de extensiones para Visual Paradigm for UML, además de estudiar las metodologías de desarrollo, lenguajes y herramientas con el fin de seleccionar las que se adecuan al proceso de evolución de la extensión.

1.4 Desarrollo de extensiones para Visual Paradigm for UML

La implementación de extensiones para un software informático constituye uno de los mecanismos para la incorporación de funcionalidades que no se previeron luego del lanzamiento de la aplicación. Este principio puede proveer al usuario de nuevas funcionalidades en la medida que estas sean identificadas con el uso y divulgación del sistema. Comúnmente se asocian la palabra extensión con plugin (fragmentos de software que interactúan con el núcleo de la aplicación informática para proporcionar una o varias funcionalidades, que en gran parte de los casos son muy específicas). Visual Paradigm es una de las herramientas CASE que más prestigio posee en la actualidad debido, a que permite de forma ágil y precisa el modelado de software, y a su vez cuenta con los medios para extender funcionalidades que no posee, dando soporte a las extensiones de la aplicación. VP provee de forma libre una interfaz de programación (API[4]), permitiendo a los desarrolladores implementar y reutilizar las clases e interfaces, desarrollando funciones agregadas para las necesidades específicas de cada usuario. (12)

El API de Visual Paradigm recomienda la utilización del lenguaje Java para la realización de extensiones y permite integrar el entorno de modelado visual con NetBeans, Además propone que es importante definir

[4] API (del inglés Application Programming Interface) o Interfaz de programación de aplicaciones (IPA)

el plugin en un archivo XML (plugin.xml), ya que es el que permitirá la integración de la extensión con la herramienta de manera visual. Este archivo incluye la información (ejemplo: identificación del plugin, proveedor y las bibliotecas necesarias), acciones personalizadas (menú, barra de herramientas y menú emergente) y formas o conectores personalizados del plugin (12).

Una vez terminado el desarrollo de la extensión, es necesario realizar el despliegue de la aplicación, es decir, ponerlo en el entorno de producción. El despliegue o expansión del plugin es muy sencillo, sólo se tiene que copiar a una carpeta específica de la aplicación cliente VP. La realización del proceso de despliegue se explica de forma detallada en el **CAPITULO 2, epígrafe 2.2**.

1.5 Metodologías de desarrollo

Las metodologías de desarrollo de software tienden a ser indispensables para la documentación correcta y eficaz de un producto de software, debido a que estas ofrecen un conjunto de procedimientos, técnicas y ayudas a la documentación necesaria para la liberación y aceptación internacional del software. Ellas muestran paso a paso las acciones y actividades a realizar para lograr el producto deseado, indicando además que personas deben participar en el progreso de las actividades y qué papel deben tener. Una metodología es capaz de continuar uno o varios modelos del ciclo de vida, detallando la información que debe obtenerse como resultado de una determinada actividad del proyecto y la información necesaria para su comienzo, indicando que productos parciales y finales hay que obtener a lo largo del desarrollo del proyecto. (13)

Entre las metodologías de desarrollo más conocidas están Programación Extrema (Extreme Programming, XP) y OpenUp, clasificadas como metodologías de desarrollo ágil o ligera, y Proceso Unificado de Desarrollo (Rational Unified Process, RUP) clasificada como metodología de desarrollo pesada.

1.5.1 Proceso Unificado de Desarrollo (Rational Unified Process, RUP)

Según la metodología de desarrollo RUP, un proceso de desarrollo de software define quién hace qué, cómo y cuándo. A su vez, precisa cuatro elementos fundamentales, trabajadores (roles) que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los artefactos (productos), que responden a la pregunta ¿Qué?, y los flujos de trabajo de las disciplinas que responden a la pregunta ¿Cuándo? (14)

RUP se caracteriza por ser dirigido por casos de usos y centrado en la arquitectura. Los casos de uso son los que guían el proceso de desarrollo y la arquitectura muestra la visión común del sistema completo. También es iterativo e incremental, proponiendo en cada fase un desarrollo en iteraciones, estas hacen referencia a pasos durante el flujo de trabajo e incrementa el crecimiento del producto final. (15)

La metodología está compuesta por cuatro fases de desarrollo de software:

- **Inicio:** donde se determina la visión y alcance del proyecto.
- **Elaboración:** donde se debe determinar una arquitectura óptima para el sistema.
- **Construcción:** donde se obtiene la capacidad operacional inicial.
- **Transición:** donde se debe obtener el release (versión operacional del producto) del proyecto. (15)

1.5.2 OpenUp

OpenUp es una variante ágil del Proceso Unificado que aplica el desarrollo iterativo e incremental dentro de la estructura del ciclo de vida. Se adopta el pragmatismo, y la filosofía ágil que se centra en la colaboración natural del desarrollo de software. Es una metodología ligera que puede ser utilizada en varios tipos de proyectos de software.

Desde la perspectiva de los stakeholders (partes interesadas) del proyecto, OpenUp estructura el ciclo de vida en 4 fases: Inicio, Elaboración, Construcción y Transición. El ciclo de vida provee la visibilidad y los puntos de decisión tanto para stakeholders como miembros del equipo, permitiendo una vigilancia efectiva del proceso de desarrollo y facilitado la toma de decisiones apropiadas en cada instante.

El ciclo de vida del proyecto provee tanto a stakeholders como a los miembros del equipo de visibilidad, de puntos de sincronización, y de puntos de decisión sobre el proyecto, hace posible la vigilancia del proyecto facilitando en cada momento la toma de decisión. (16)

1.5.3 Programación Extrema (Extreme Programming, XP)

La metodología de desarrollo Programación Extrema (XP), se ha convertido en la actualidad en una de las metodologías más exitosas, pues cuenta con la aceptación de la mayoría de los equipos de desarrollo de software. Su autor Kent Beck la definió como una metodología ligera para la creación de software debido a su simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Su

característica más sobresaliente se encuentra en que el cliente forma parte del equipo de desarrollo, requisito necesario para alcanzar el éxito del proyecto. (17)

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva de proceso. El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierten en miembro del equipo. (17)

Lo fundamental en este tipo de metodología es:

- La comunicación entre los usuarios y los desarrolladores.
- La simplicidad al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales. (17)

La metodología XP se divide en 4 fases:

1. **Planificación:** plantea la planificación como un permanente diálogo entre las partes, la empresarial (deseable) y la técnica (posible).
2. **Diseño:** plantea que el diseño adecuado para el software es el que funciona con todas las pruebas, no tiene lógica duplicada, manifiesta cada intención importante para los programadores, tiene el menor número de clases y métodos.
3. **Desarrollo:** plantea la recodificación, programación por parejas, la propiedad colectiva, integración continua del código.
4. **Pruebas:** plantea que no debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para chequear el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. (18)

Roles y artefactos

Un rol es una definición abstracta de un conjunto de actividades realizadas y de artefactos obtenidos. Los roles son realizados típicamente por un individuo, o un conjunto de individuos, trabajando juntos en equipo. Un miembro del equipo de desarrollo del proyecto puede cumplir normalmente muchos roles o un único rol. Los roles no son individuos, sino responsabilidades que deben asumir las personas en un determinado proyecto y describen cómo estos se comportan en el negocio. (19)

De acuerdo con las necesidades y características de esta investigación, los roles definidos para el desarrollo de la solución serán: programador, cliente y encargado de pruebas.

1. **Programador:** en la metodología XP los programadores diseñan, programan y realizan pruebas. Son los responsables de tomar decisiones técnicas y construir el sistema. Los artefactos principales serán elaborados por él.

Artefactos generados por el rol de programador:

- **Tarjetas CRC:** el uso de tarjetas CRC (Clase, Responsabilidades y Colaboraciones) sirven para diseñar el sistema en conjunto entre todo el equipo. Estas tarjetas representan objetos, la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha las clases que colaboran con cada responsabilidad. (16)

2. **Cliente:** es parte del equipo, determina qué construir y puede establecer pruebas funcionales. (20)

Artefactos generados por el rol de cliente:

- **Historias de usuario:** tienen el mismo propósito que los casos de uso. Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema. Existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conlleva la implementación de dicha historia de usuario. (16)

3. **Encargado de pruebas:** es el responsable de ayudar al cliente a escoger y escribir pruebas funcionales. Es responsable también de hacer funcionar las pruebas regularmente y comunicar sus resultados al resto del equipo. Debe ser una persona con capacidad de abstracción y mucho tacto

y facilidad de comunicación, dado que está en un lugar intermedio entre los programadores y el cliente. (20)

Artefactos generados por el rol de encargado de pruebas:

- **Casos de prueba**: es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados, para cumplir un objetivo en particular o una función esperada. La entidad más simple que siempre es ejecutada como una unidad, desde el comienzo hasta el final. (16)

1.6 Selección de la metodología

Es cierto que las metodologías de desarrollo de software tienden a ser indispensables para la documentación correcta de un producto de software. Pero si la misma no se escoge correctamente, puede tener consecuencias sobre el costo y tiempo de entrega del producto. Para evitar contratiempos de este tipo, se realizó un estudio de las características del equipo de desarrollo y del producto que se desea implementar. El principal objetivo de este análisis, es la correcta selección de la metodología que se utilizará en todo el ciclo de vida del proyecto.

Se seleccionó la metodología XP, la descripción de la misma es aplicable a las características específicas del proyecto general y en aras de simplificar el desarrollo de software y reducir en tiempo y costo, los esfuerzos invertidos en el producto. Es mucho más fácil de implementar y de aprender, por lo que el equipo puede incorporarla de manera más natural. Dentro de las características específicas del proyecto están las particularidades del equipo de desarrollo: es pequeño, los programadores son comunes (poseen conocimientos básicos) y el cliente (grupo de profesores del departamento), está integrado de manera natural en el ámbito de trabajo. Por su parte el proyecto es a corto plazo, ya que se desea concebir en un período de tiempo pequeño. La metodología XP combina las que han demostrado ser prácticas sobresalientes en la industria de desarrollo de software. La metodología permitirá todo el tiempo rediseñar el código generado (refactorización), dejando el código siempre en el estado más simple posible. Se hacen pruebas unitarias continuas, frecuentemente repetidas, automatizadas y funcionales donde los clientes comprobaran que el proyecto va satisfaciendo los requisitos. Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, esto permite discusiones diarias informales, para fomentar la comunicación, y para hacer que los desarrolladores tengan tiempo de hablar de los problemas a los que se enfrentan y de ver cómo van con sus trabajos.

1.7 Lenguaje de programación y de marcas

Un lenguaje de programación describe un conjunto de acciones que un equipo debe ejecutar. Está conformado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al conjunto de instrucciones que se genera se conoce como código fuente de un programa. El lenguaje de programación permite a un programador especificar sobre qué datos la computadora debe operar, cómo deben ser almacenados y transmitidos y cuáles acciones ejecutar ante determinadas circunstancias. (21)

1.7.1 Java

Entre los diversos lenguajes de programación se encuentra Java que es un lenguaje que emplea el paradigma de programación orientado a objeto para propósito general. Es descendiente de un lenguaje llamado Oak cuya intención era la creación de software para la televisión interactiva. Consta de plataforma independiente y fue desarrollado por Sun Microsystems en los años 90. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel. Su principal característica es la de ser un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera es interpretado por una máquina virtual, de este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma. (21)

Para la implementación de la extensión se empleó este lenguaje, por ser el lenguaje de programación propuesto por el API de desarrollo de la herramienta VP.

1.7.2 Extensible Markup Language (XML)

XML (Lenguaje de Marcas Extensible) es un metalenguaje (un lenguaje que se utiliza para decir algo sobre otro lenguaje). Fue desarrollado por el consorcio web (W3C) en 1995 que es un consorcio internacional que elabora recomendaciones para la red informática mundial (WWW). (10)

El XML es una adaptación del SGML (Standard Generalized Markup Language), un lenguaje que permite la organización y el etiquetado de documentos. Esto quiere decir que el XML no es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades. El XHTML (eXtensible Hyper Text Markup Language), el MathML (Mathematical Markup Language) y el SVG (Scalable Vector Graphics) son algunos de los lenguajes que el XML está en condiciones de definir. (10)

Las bases de datos, los documentos de texto, las hojas de cálculo y las páginas web son algunos de los campos de aplicación del XML. El metalenguaje aparece como un estándar que estructura el intercambio de información entre las diferentes plataformas.

Los expertos nombran varias ventajas que derivan de la utilización del XML: (22)

- Es extensible (se pueden añadir nuevas etiquetas tras el diseño del documento).
- Su analizador es estándar (no requiere de cambios para cada versión del metalenguaje) y facilita el análisis y procesamiento de los documentos XML creados por terceros.
- Numerosas herramientas de procesamiento.
- Admite la definición de vocabularios específicos.
- Separa contenido del procesamiento y visualización.
- Aumenta la seguridad mediante la validación de documentos.
- Formato abierto, respaldado por numerosas organizaciones.

El API de desarrollo de la herramienta VP, propone que es de vital importancia definir el XML, ya que es el que permitirá la integración de la extensión con la herramienta de manera visual.

Una vez definido los lenguajes de programación, es necesario tener en cuenta las herramientas con las que se va a realizar el diseño y la implementación del plugin. A continuación se muestran las características de las herramientas más afines con la solución.

1.8 Herramientas para el desarrollo

Cuando se pretende definir las herramientas a utilizar en la implementación de un proyecto, es importante tener en cuenta que se debe elegir adecuadamente la tecnología que cada empresa necesita o tiene al alcance según posibilidades y características. Por este motivo es necesario investigar el mercado de aplicaciones para la industria y la empresa, con el fin de lograr una selección acorde a las necesidades del producto y la institución.

1.8.1 Visual Paradigm for UML 8.0

Visual Paradigm for UML (Unified Modeling Language) es una herramienta CASE, multiplataforma de modelado UML, muy potente y fácil de utilizar. Posee una licencia Freeware (software gratis). Aporta a los desarrolladores de software una plataforma de desarrollo para construir aplicaciones de gran calidad y

rapidez. Permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML y generar código fuente desde dichos diagramas. (23)

Soporta el ciclo de vida completo de desarrollo de un software, desde la fase de análisis hasta el despliegue del mismo. Además de la generación automática de informes en formato PDF, Word o HTML. Esta herramienta además se puede integrar con diversos IDE[5] como NetBeans (de Sun), Developer (de Oracle), Eclipse (de IBM), JBuilder (de Borland). Es fácil de instalar, actualizar y es compatible entre ediciones. Entre sus principales características se evidencian: (23)

- Editor de Detalles de Casos de Uso: entorno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Interoperabilidad: intercambia diagramas UML y modelos con otras herramientas. Soporta la importación y exportación a formatos XML y archivos Excel.
- Colaboración de Equipo: realiza el modelado simultáneamente con el Paradigm TeamWork Server y Subversión.
- Modelado de Requisitos: captura de requisitos mediante diagramas de requisitos, modelado de caso de uso y análisis textual.
- Multiplataforma: soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
- Generación de Documentación: comparte y genera documentación de diagramas y diseños en formatos PDF, HTML, Microsoft Work.
- Ingeniería de Código: permite la generación de código e ingeniería inversa para los lenguajes: Java, C, C++, PHP, XML, Python, C#, VB .Net, Flash, ActionScript, Delphi y Perl.
- Ingeniería Inversa: código a modelo, código a diagrama.
- Integración con Entornos de Desarrollo: apoyo al ciclo de vida completo de desarrollo de software en IDEs como: Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper y Jbuilder.
- Modelado de Bases de datos**:** generación de bases de datos y conversiones de diagramas entidad-relación a tablas de bases de datos, además de mapeos de objetos y relaciones.

[5] IDE (sigla en inglés de Integrated Development Environment) o entorno de desarrollo integrado

Se utiliza esta herramienta por sus facilidades y por ser la escogida en la UCI para el desarrollo de software, siguiendo las indicaciones propuestas por el país para obtener la soberanía tecnológica en breve tiempo haciendo uso de las políticas de software libre.

Lenguaje de Modelado

El lenguaje de modelado está formado por un conjunto de símbolos que permitirán modelar parte de un diseño de software orientado a objetos. Se utiliza extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para mostrar y comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo. En la informática, como en las demás ramas de la ingeniería se necesita organizar, planificar y diseñar el trabajo antes de ejecutarlo, para así reducir las probabilidades de error en la ejecución del mismo. Los diseños se entienden mejor si se representan gráficamente, por lo que surgió la necesidad de representar los diseños del software que se fueran a desarrollar. Para poder estandarizar estos diseños y que fueran entendibles para todos, surge UML, dando así, paso a una de las premisas de la ingeniería del software, la reutilización. (24)

Lenguaje Unificado de Modelado (Unified Modeling Language o UML)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como: procesos de negocio y funciones del sistema. Además, presenta aspectos concretos como: expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

UML es además un método formal de modelado, lo que aporta las siguientes ventajas: (25)

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar códigos a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el

código estén actualizados, con lo que siempre se puede mantener la visión en el diseño de más alto nivel de la estructura de un proyecto.

Los conceptos de modelado de UML están agrupados en unidades de lenguaje (Language Unit). Una unidad de lenguaje consiste en una colección de conceptos fuertemente emparejados, que proveen a los usuarios con el poder de representar aspectos del sistema a estudiar de acuerdo a un paradigma en particular. Desde la perspectiva del usuario esta partición de UML significa que el usuario solamente necesita preocuparse por aquellas partes que considere necesarias para el modelado. Por tanto, un usuario de UML no necesita saber el lenguaje completo para usar UML efectivamente. (24)

En este trabajo se ha profundizado en el estudio UML por la necesidad de interpretar modelos que van a estar estructurados a partir de este lenguaje, pero es importante tener en cuenta que en ningún momento se realiza un modelo UML.

1.8.2 IDE NetBeans 7.1

El IDE NetBeans 7.1 es un entorno integrado de desarrollo de software muy aceptado dentro de la comunidad internacional de programadores. El IDE se encuentra disponible para Windows, Mac, Linux y Solaris. Este IDE es gratuito y de código abierto. Consta con una plataforma de varias aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, escritorio y aplicaciones móviles utilizando la plataforma Java, así como, JavaFX, PHP 5.3, Java Script y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C/C++. NetBeans IDE 7.1 continua dando soporte para el compositor de JavaFX ahora con la versión JavaFX 2.0, la cual es una herramienta de diseño visual para construir visualmente aplicaciones JavaFX interfaz gráfica de usuario, similar a la del constructor Swing GUI para aplicaciones Java SE con el compositor de JavaFX. Los desarrolladores en el compositor de JavaFX 2.0 pueden crear rápidamente, visualmente editar y depurar aplicaciones dinámicas de Internet (RIA) y los componentes se unen a diversas fuentes de datos, incluidos los servicios Web. Ofrece entre sus mejoras: (12)

- jdk7 (software que provee herramientas de desarrollo para la creación de programas en java) y el autocompletado de código y pistas.
- Integración con JSF y librerías del servidor.
- Integración con JUnit 4.8.2 y varias mejoras en JUnit (entorno que permite ejecutar tests de clases Java).

- Refactoring (refactorización) de renombrado y de borrado seguro.

Y de forma general:

- Wordwrap (ajuste automático de líneas) en el editor.
- Mejorada la integración del Profiler.

La comprobación de cambios externos en ficheros ahora es menos intrusiva, cuando se cambia de tarea entre el IDE y otros programas.

Entorno de desarrollo

Se asume como herramienta CASE ¨Visual Paradigm for UML¨ por sus facilidades y por ser la escogida en la UCI para el desarrollo de software. Como IDE, el NetBeans 7.1, dada las características y comodidades que ofrece, además es un entorno de desarrollo moderno y potente en el que se pueden crear distintos tipos de aplicaciones Java, siendo este el lenguaje recomendado por el API de desarrollo de VP.

Conclusiones parciales

A partir de las necesidades del departamento Almacenes de datos se determinó la implementación del plugin para la herramienta de modelado Visual Paradigm for UML en su versión 8.0, ya que satisface las condiciones de desarrollo de la extensión y por estar contenido en el entorno tecnológico de DATEC. Los orígenes de datos, que van a ser manejados en el diseño y confección del Mapa Lógico de Datos son las Hojas de cálculo, los ficheros XML y la conexión a las fuentes de datos mediante los Gestores de Base de Datos PostgreSQL y MySQL, además los ficheros sql generados por estos gestores. Estas fuentes de datos se escogen, ya que, son las que se utilizan en el departamento. Las metodologías RUP y OpenUp no se corresponden a las necesidades de la extensión, la que se adecua a las características del proyecto y al equipo de desarrollo es XP y se escoge como metodología de desarrollo de software. Se emplea como IDE NetBeans 7.1 y Java como lenguaje de programación, ya que es el propuesto por el API de desarrollo de la herramienta VP.

CAPÍTULO 2. ANÁLISIS Y DISEÑO DE LA EXTENSIÓN PARA LA ELABORACIÓN DEL MAPA LÓGICO DE DATOS

En este capítulo se presentará la propuesta de la extensión para la gestión del Mapa Lógico de Datos, con el objetivo de dar solución a la situación problemática anteriormente planteada. Se definirán las principales características que poseerá la aplicación informática a desarrollar. Se realizará una especificación de los requisitos funcionales, no funcionales y el diseño. Además, se realizarán los artefactos correspondientes a estas fases según el ciclo de vida que propone la metodología seleccionada, para lograr un mejor entendimiento del componente.

2.1 Propuesta de solución a la problemática planteada

La integración de datos en el Departamento Almacenes de Datos, es el proceso que más tiempo y esfuerzo requiere en la construcción de un almacenen de datos. Un elemento esencial para el diseño del subsistema de integración de datos es la confección del Mapa Lógico de Datos. Los retrasos que genera la realización de este artefacto afectan considerablemente los tiempos de desarrollo de la solución en general. Por este motivo sepropone realizar una extensión para la herramienta “Visual Paradigm for UML” que permita gestionar el Mapa Lógico de Datos.

El elemento fundamental de la extensión es relacionar el origen y el destino de los datos, por lo que se hace necesario que el plugin sea capaz de cargar la estructura del almacén de datos y los ficheros con las diferentes fuentes de datos (ficheros xml, xls y los ficheros sql generados por los gestores de base de datos PostgreSQL o MySQL) que utiliza el departamento, además del manejo de los mismos, permitiendo establecer relaciones entre las tablas y atributos de la fuente y el destino. El sistema deberá permitir cargar, exportar y actualizar el Mapa Lógico de Datos.

Cuando se trabaja con extensiones para la herramienta CASE “Visual Paradigm for UML”, es necesario dominar la estructura de desarrollo y conocer un grupo de pasos, que guiarán un correcto desarrollo del proceso integración del plugin. Estos pasos se detallan a continuación:

2.2 Pasos para desarrollar una extensión para la herramienta CASE "Visual Paradigm for UML"

Para extender o crear una extensión para VP es importante tener conocimiento de la estructura de desarrollo, así como la posterior integración de la extensión con la herramienta CASE. Es por ello que para su correcto desarrollo se realizó la siguiente consecución de pasos:

Primer paso: La herramienta VP ofrece un medio para la extensión de la aplicación mediante la librería openapi.jar, que se encuentra ubicada dentro de los paquetes de instalación de la herramienta, específicamente en el paquete "lib/openapi.jar". Luego de conocer donde se encuentra la librería para la construcción de la extensión, es necesario tener en cuenta la estructura que debe tener la extensión, que para el caso del IDE de desarrollo NetBeans tiene la estructura de un paquete que lleva el nombre de la extensión, y contendrá a su vez tres paquetes asociados relativos a:

1. Configuración de la extensión.
2. Acciones correspondientes.
3. Formularios o diálogos de la implementación.

De cada uno de estos paquetes se debe dominar su funcionamiento y qué relación establecen con el openapi.jar incorporado al proyecto. El primer paquete está asociado a la configuración de la extensión y es necesario conocer que se encuentra formado por las clases Plugin.xml y Plugin.java, las cuales permiten la carga y configuración del mismo.

Implementación de Plugin.xml y Plugin.java

Esta permite mediante un Script xml definido, la configuración de la extensión que va a ser cargado por la clase Plugin.java a través de la implementación de la interfaz VPPlugin ofrecida por la librería openapi.jar, la cual implementa los métodos load y unload, habilitando la carga y descarga de la extensión. Esta permite la conexión entre las librerías asociadas a la implementación, así como las configuraciones de las acciones tanto a nivel de herramienta como a nivel de contexto cargando las clases correspondiente a dicha acción. Ejemplo de la implementación del archivo Plugin.xml ver **Anexo 1**

Ejemplo de implementación de la clase Plugin.java

```
public class Plugin implements com.vp.plugin.VPPlugin {
// Asegurarse de que el constructor de la clase no tenga parámetros
```

```
  public void loaded (com.vp.plugin.VPPluginInfo info) {
// Llamado cuando el plugin es cargado
   }
   public void unloaded () {
// Llamado cuando el plugin es descargado
    }
}
```

El segundo paquete contiene las acciones correspondientes a realizar por la extensión, las cuales se encuentran definidas a nivel de herramienta y de contexto. Estas clases, en dependencia de cuál sea la acción, implementan la interfaz asociada a dicha acción, VPActionController y VPContextActionController. Dichas clases especifican el performAction pudiendo así ejecutar acciones referentes al evento onClick de la acción. Ejemplo de la implementación de las clases VPActionController y VPContextActionController ver **Anexo 2**.

Mientras que el tercer paquete del proyecto contiene los diálogos que se necesitan mostrar a través del método performAction, que está asociado a la acción correspondiente al dialogo que se necesita utilizar.

Al culminar con la implementación y estructuración del proyecto, se está en condiciones de desarrollar una plantilla de extensión para la herramienta CASE VP. A continuación la **Figura 2** muestra la estructura de la plantilla de la extensión.

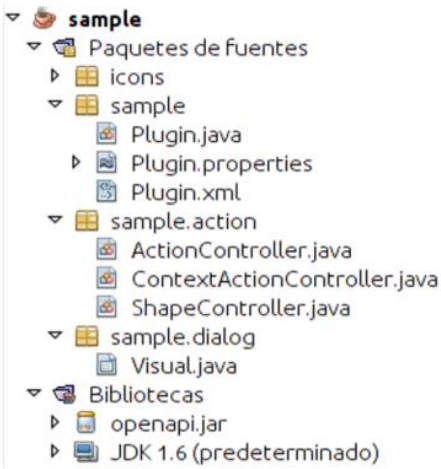


Figura 2 Estructura de un proyecto de extensión para "Visual Paradigm for UML" en el IDE NetBeans

Segundo paso: integración de la extensión con la herramienta CASE VP. Para dicha integración VP propone una estructura a seguir en el despliegue de la extensión, la cual que se muestra en la **Figura 3**. Hay que crear una carpeta con el nombre plugin dentro de los paquetes de instalación de la aplicación, es decir, buscar dónde se encuentra instalado VP y crear la estructura de paquete de despliegue: (26)

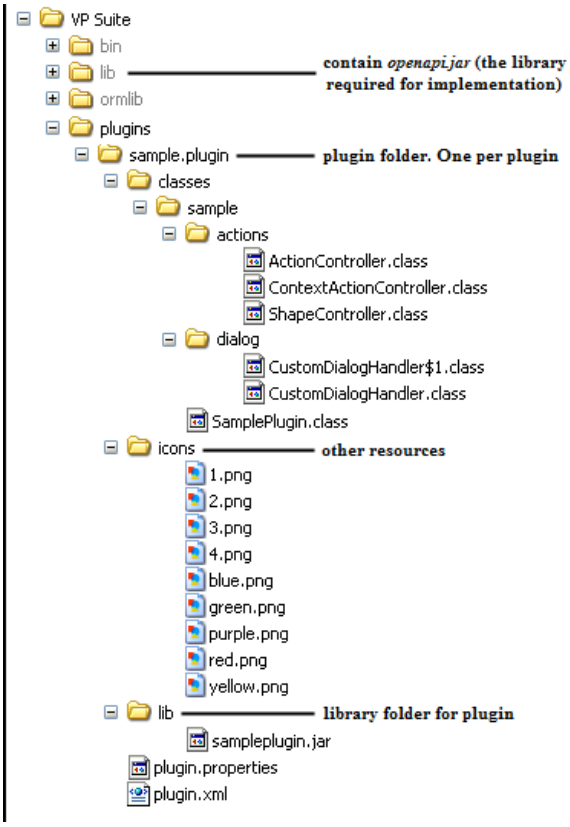


Figura 3 Estructura de despliegue de la extensión

Si se observa dicha estructura de implementación de la extensión, difiere significativamente con respecto a la estructura de despliegue para la herramienta, lo cual puede dificultar el proceso de pruebas a la hora de integrar la extensión desarrollada con VP. Es por ello que para evitar dichos problemas se utiliza una herramienta de despliegue para la extensión, que facilite la integración del mismo con VP, ver **Figura 4**. La aplicación hace uso de tres argumentos:

1. En el primer paso: Nombre del plugin o extensión.
2. En el segundo: Dirección origen
3. En el tercero: Dirección destino

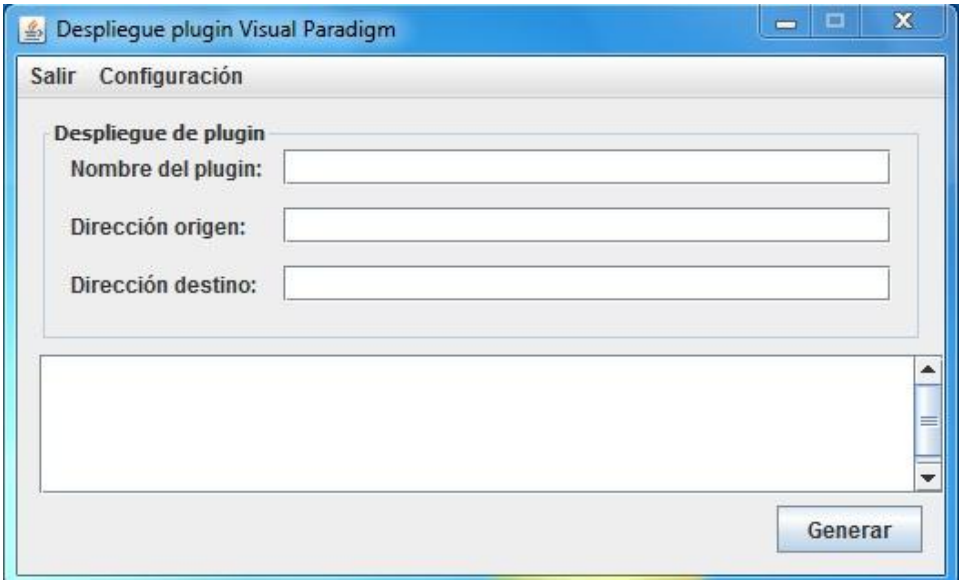


Figura 4 Interfaz del despliegue de la extensión

Luego de detallar la estructura de desarrollo y los pasos para la integración del plugin, se puede definir la fase de planificación que es la primera fase de desarrollo planteada por la metodología XP para el desarrollo del sistema.

2.3 Fase de planificación del sistema

La metodología XP asume que la planificación nunca podrá ser perfecta, debido a que varía en función de las necesidades del negocio. En esta fase se definen los requerimientos funcionales y no funcionales que tendrá el sistema, a partir de las necesidades de los clientes que están representados por un grupo de profesores del departamento. Se realiza la Lista de Reserva del Producto para tener una estimación y control detallado de las funcionalidades que se van a implementar. Se establece la prioridad de cada historia de usuario, y correspondientemente, se realiza una estimación del esfuerzo necesario de cada una de ellas. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores 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, el equipo de desarrollo 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.

2.3.1 Requerimientos funcionales

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Definen las funciones que el sistema será capaz de realizar. Expresan la naturaleza del funcionamiento del sistema, cómo interactúa el sistema con su entorno y cuáles van a ser su estado y funcionamiento.

Para el desarrollo de este sistema se han definido los siguientes requisitos funcionales de acuerdo a las características que presenta la extensión: (27)

- **RF1** Cargar ficheros con extensión xml.
- **RF2** Cargar los ficheros sql generado por los gestores de base de datos PostgreSQL o MySQL.
- **RF3** Cargar ficheros con extensión xls.
- **RF4** Conectarse a las fuentes de datos mediante los gestores de base de datos PostgreSQL o MySQL.
- **RF5** Relacionar la fuente y el destino de los datos a partir los ficheros cargados o mediante la conexión a la base de datos.
- **RF6** Guardar el Mapa Lógico de Datos.
- **RF7** Exportar el Mapa Lógico de Datos en formato pdf.
- **RF8** Actualizar el Mapa Lógico de Datos.

2.3.2 Requerimientos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. (28)

Los requerimientos no funcionales identificados son:

- **RNF1** Requerimientos de software

  Sistema operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 10.4 o superior, Debían 6.0 GNU/Linux o superior, Windows XP o superior.

  Máquina Virtual de Java JDK y JRE tanto para GNU/Linux como para Windows.

- **RNF2** Restricciones del diseño y la implementación

Será una extensión para VP, para el desarrollo de la misma se emplea la metodología XP.

Se utiliza Java como lenguaje de programación.

- **RNF3** Requisitos de soporte

Proveer un manual de usuario.

- **RNF4** Requisitos de portabilidad

Será un sistema multiplataforma, lo que permitirá poder disponer del mismo en cualquier sistema operativo.

- **RNF5** Apariencia o interfaz externa.

La extensión al integrarse con la herramienta VP, tendrá una interfaz amigable y fácil de interactuar para hacer más factible el uso para los usuarios.

2.3.3 Modelo de dominio

En la metodología XP no está definido una técnica para determinar el negocio, por lo que es necesario realizar este proceso de una manera entendible para llevar a cabo el desarrollo de la extensión. Una de las técnicas más utilizadas es el modelo de dominio, ayudando a facilitar el entendimiento del software que se desea desarrollar. Es considerado uno de los artefactos necesarios que debe contener cualquier proyecto. El Modelo de dominio es una representación visual estática del entorno, un diagrama con los objetos que existen (reales) relacionados con el proyecto. Ayuda a comprender los conceptos que utilizan los usuarios, conceptos con los que trabajan y con los que deberá trabajar la aplicación (29).La **Figura 5** muestra el modelo de dominio que representa el sistema que se desarrollará.

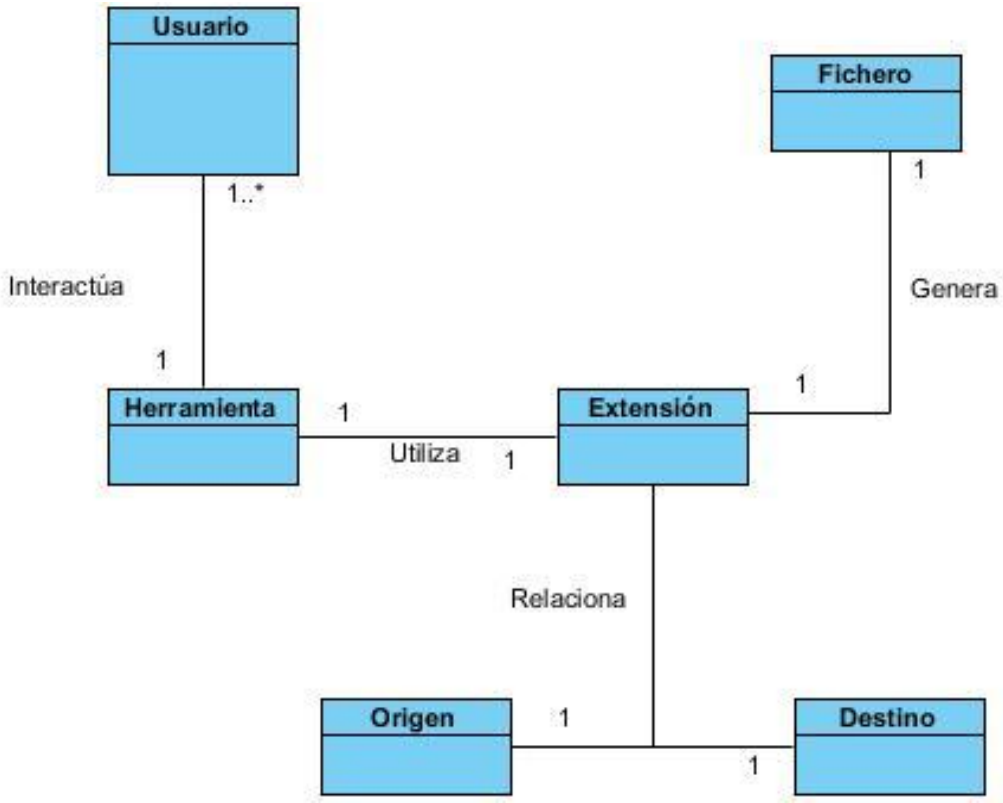


Figura 5 Modelo de dominio

Tabla 1 Definición de los conceptos del modelo de dominio

Elementos	Descripción
Usuario	Persona facultada para utilizar el plugin asistido a la herramienta “Visual Paradigm”.
Herramienta	Herramienta CASE “Visual Paradigm”.
Extensión	Plugin integrado a la herramienta “Visual Paradigm”.
Origen	Ficheros de distintos tipos que serán cargados por la herramienta.
Destino	Almacén de datos cargado por la herramienta.
Fichero	Mapa Lógico de Datos que se genera, el mismo se podrá guardar, exportar y actualizar.

2.3.4 Lista de reserva del producto

La Lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir la aplicación que se desea realizar, ordenados según la prioridad de implementación, ubicados en muy alta, alta, media y baja, en esta última aparecen los requisitos de menor complejidad, además de los requisitos

no funcionales (es opcional ponerlos en la tabla) del sistema a desarrollar. Indica la estimación de cada uno de ellos y su implementación por semanas y quien hizo la estimación.

Tabla 2 Lista de reserva del producto

Ítem	Descripción	Estimación	Estimado
Prioridad: Muy Alta			
1	Cargar ficheros con extensión xml	9	Programador
2	Cargar los ficheros sql generado por los gestores de base de datos PostgreSQL o MySQL		
3	Cargar ficheros con extensión xls		
4	Conectarse a las fuentes de datos mediante los gestores de base de datos PostgreSQL o MySQL		
5	Relacionar la fuente y el destino de los datos a partir los ficheros cargados		
Prioridad: Alta			
6	Guardar el Mapa Lógico de Datos	3,5	Programador
7	Exportar el Mapa Lógico de Datos a partir de diferentes fuentes de datos		
Prioridad: Media			
8	Actualizar el Mapa Lógico de Datos	1	Programador

2.3.5 Historias de usuarios

Las historias de usuarios son artefactos generados en la metodología XP, tienen la misma finalidad que los casos de uso, pero con algunas diferencias: constan de tres o cuatro líneas escritas por el cliente en un lenguaje no técnico, no se debe hablar de posibles algoritmos para su implementación. Son una técnica para especificar requisitos y son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Para el presente trabajo se realizaron un total de 6 historias de usuarios. A continuación se muestran las historias de usuario definidas para la implementación de la extensión.

Tabla 3 HU 1 ¨Cargar los ficheros desde la fuente¨

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Cargar los ficheros desde la fuente
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 3 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 3 semanas
Descripción: Carga los ficheros fuente permitiéndole al usuario buscarlo en la dirección guardada	
Observaciones:	

Tabla 4HU 2 ¨Cargar la estructura del almacén de datos¨

Historia de Usuario	
Número: 2	**Nombre de la Historia de Usuario:** Cargar la estructura del almacén de datos
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 3 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 3 semanas
Descripción: Carga la estructura del almacén permitiéndole al usuario buscarlo en la dirección guardada	
Observaciones:	

Tabla 5 HU 3 ¨Relacionar la fuente y el destino de los datos¨

Historia de Usuario	
Número:3	**Nombre de la Historia de Usuario:** Relacionar la fuente y el destino de los datos

Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 3
Prioridad en negocio: Alta	**Puntos estimados:** 3 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 3 semanas
Descripción: Permite relacionar las tablas y atributos de la fuente con el almacén	
Observaciones:	

Tabla 6 HU 4¨Guardar el Mapa Lógico de Datos¨

Historia de Usuario	
Número:4	**Nombre de la Historia de Usuario:** Guardar el Mapa Lógico de Datos
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 4
Prioridad en negocio: Medio	**Puntos estimados:**1.5 semana
Riesgo en desarrollo: Medio	**Puntos reales:** 1.5 semana
Descripción: Permite guardar en un destino especificado el Mapa Lógico de Datos	
Observaciones:	

Tabla 7 HU 5 ¨Actualizar el Mapa Lógico de Datos¨

Historia de Usuario	
Número:5	**Nombre de la Historia de Usuario:** Actualizar el Mapa Lógico de Datos
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 4
Prioridad en negocio: Baja	**Puntos estimados:**1 semana
Riesgo en desarrollo: Bajo	**Puntos reales:** 1 semana

Descripción: Permite cargar y actualizar el Mapa Lógico de Datos
Observaciones:

Tabla 8 HU 6 ¨Exportar el Mapa Lógico de Datos en formato pdf˙

Historia de Usuario	
Número:6	**Nombre de la Historia de Usuario:** Exportar el Mapa Lógico de Datos en formato pdf
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Liena Calzado Robaina	**Iteración asignada:** 5
Prioridad en negocio: Media	**Puntos estimados:**2 semana
Riesgo en desarrollo: Medio	**Puntos reales:** 2 semana
Descripción: Brinda la posibilidad de exportar el Mapa Lógico de Datos en formato pdf luego de relacionar la fuente y el destino	
Observaciones:	

2.3.6 Plan de iteraciones

Luego de tener definidas las historias de usuarios se debe crear un plan de iteraciones o plan de release, indicando cuáles se desarrollarán en cada iteración del programa. El objetivo de este artefacto es tener una planificación del trabajo, donde los desarrolladores y el cliente establecen los tiempos de implementación de las historias de usuarios y la prioridad con que serán desarrolladas.

Tabla 9 Plan de iteraciones

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	En esta iteración se implementará la historia de usuario que se encargará del proceso de cargar las fuentes de datos	1	3 semanas
2	En esta iteración se implementará la historia de usuario que se encargará de cargar la estructura del almacén de datos	2	3 semanas
3	En esta iteración se implementará la historia de	3	3 semanas

	usuario que se encargará de relacionar la fuente y el destino		
4	En esta iteración se implementarán las historias de usuarios correspondientes a guardar y actualizar el Mapa Lógico de Datos	4-5	2,5 semanas
5	En esta iteración se implementará la historia de usuario correspondientes a exportar el Mapa Lógico de Datos	6	2 semanas

2.4 Fase de diseño del sistema

En el desarrollo de la extensión es importante un adecuado diseño de las clases que la componen, para realizarlo con la mejor calidad posible, y así el cliente quede satisfecho. La metodología X.P sugiere que hay que conseguir diseños simples y sencillos, procurando hacerlo todo lo menos complicado posible para conseguir un diseño que permita ahorrar tiempo y esfuerzo para desarrollar la solución. Para ello la metodología se apoya en las tarjetas CRC, las cuales permiten al programador centrarse y apreciar el desarrollo orientado a objetos, olvidándose de los malos hábitos de la programación estructurada clásica.

2.4.1 Tarjetas CRC

Las tarjetas CRC son una técnica simple e informal pero efectiva, que ha sido propuesta tanto para el modelado conceptual como para el diseño detallado de sistemas. Una tarjeta CRC establece 3 dimensiones las cuales identifican el rol de un objeto en análisis y/o diseño: nombre de la clase, responsabilidades y colaboraciones.

Estas tarjetas sirven para diseñar el sistema en conjunto con todo el equipo. Permiten reducir el modo de pensar y apreciar la tecnología de objetos, el uso de estos diagramas puede aplicarse siempre, no como un artefacto de la metodología, pero se pueden utilizar siempre y cuando influyan en el mejoramiento de la comunicación, no será un peso su mantenimiento, no serán extensos y se enfocan en la información importante.

Para el presente trabajo se obtuvo un total de 14 tarjetas CRC. A continuación se muestran 4 de las tarjetas CRC definidas para la extensión. El resto está ubicado en el expediente de proyecto de la solución.

Tabla 10 Tarjeta CRC ¨Clase: Leer_XML¨

Tarjeta CRC	
Clase: Leer_XML	
Responsabilidades	**Colaboraciones**
Cargar tablas Leer tablas	Tabla

Tabla 11 Tarjeta CRC ¨Clase: Leer_Base_Datos¨

Tarjeta CRC	
Clase: Leer_Base_Datos	
Responsabilidades	**Responsabilidades**
Obtener el nombre de la base datos Obtener las clases Obtener los atributos y su tipo de dato dado una clase Identificar el tipo de estereotipo de las clases	Obtener el nombre de la base datos Obtener las clases Obtener los atributos y su tipo de dato dado una clase Identificar el tipo de estereotipo de las clases

Tabla 12 Tarjeta CRC ¨Clase: Leer_Postgres¨

Tarjeta CRC	
Clase: Leer_Postgres	
Responsabilidades	**Colaboraciones**
Obtener los nombres de las tablas Obtener los atributos y tipos de dato	Conexion

Tabla 13 Tarjeta CRC ¨Cargar_BD_Fuente¨

Tarjeta CRC	
Clase: Cargar_BD_Fuente	
Responsabilidades	**Colaboraciones**
Identificar si es un diagrama de clases el que está activo Enviar la acción para mostrar el diálogo	Funciones_BD Leer_Base_Datos VPActionController

que establece la conexión en la fuente	ViewManager

Los requisitos de software evolucionan continuamente, las tecnologías que los soportan y sus funcionalidades están en constante modificación ante las necesidades crecientes de los usuarios. Ante esta realidad para los diseñadores se establece un gran reto: ¿Cómo definir un software capaz de adaptarse a estos cambios y a la ves poder reutilizar sus capacidades ya implementadas? La respuesta, un uso adecuado de patrones de diseño; estos están orientados al cambio, facilitan la reutilización de diseños y arquitecturas de software exitosas, garantizando los mecanismos necesarios para su implementación futura.

2.5 Patrones de diseño

El uso de patrones, según el arquitecto Christopher Alexander, constituye una base en la búsqueda de soluciones a problemas que surgen durante el desarrollo de software: *"Cada patrón describe un problema que ocurre una y otra vez en el 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. Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software".* (30)

Esta definición es similar a la dada por Richard Gabriel: *"Cada patrón es una regla de tres partes, la cual expresa una relación entre un cierto contexto, un conjunto de fuerzas que ocurren repetidamente en ese contexto y una cierta configuración software que permite a estas fuerzas resolverse por sí mismas".* (31)

El objetivo de los patrones de diseño es guardar la experiencia en diseños de programas orientados a objetos. Un patrón nombra, abstrae e identifica los aspectos clave de un diseño estructurado, que lo hace útil para la creación de diseños orientados a objetos reutilizables. Los patrones de diseño identifican las clases participantes y las instancias, sus papeles y colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se enfoca sobre una situación en particular. Un algoritmo puede ser un ejemplo de implementación de un patrón, pero es demasiado incompleto, específico y rígido para ser un patrón. Una regla o heurística puede participar en los efectos de un patrón, pero un patrón es mucho más. Los patrones de diseños con descripciones de las comunicaciones de objetos y clases que son personalizadas para resolver un problema general de diseño en un contexto particular. Se pueden utilizar en cualquier lenguaje de programación orientado a objetos, adaptando los diseños generales a las características de la

implementación particular. Describen el problema en forma sencilla, el contexto en que ocurre, los pasos a seguir, los puntos fuertes y débiles de la solución y si existen otros patrones asociados.

Teniendo en cuenta el estudio de patrones de diseños ya existentes y algunos contextos similares en la implementación de clases y funcionalidades específicas del sistema, se utilizarán los siguientes patrones de diseño en el desarrollo de la solución con el fin de agilizar el proceso de implementación al desarrollador:

2.5.1 Patrones de diseño GRASP [6]

Son patrones que resaltan la importancia de captar principios para la programación, a la hora de diseñar competentemente un software orientado a objetos. Los patrones GRASP describen la asignación de responsabilidades a objetos, expresados en forma de patrones.

Experto: es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Beneficios:

Se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener. Se distribuye el comportamiento entre las clases que contienen la información requerida, por tanto, se estimula las definiciones de clases más cohesivas y "ligeras" que son más fáciles de entender y mantener. Se soporta normalmente una alta cohesión (32). La **Figura 6** muestra un ejemplo del empleo de éste patrón en el sistema.

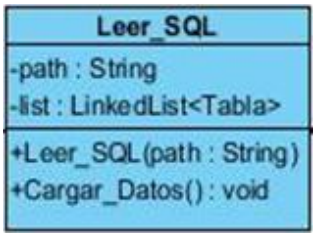


Figura 6 Ejemplo del patrón Experto

[6] Acrónimo de General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignación de Responsabilidades)

Creador: propone que un objeto tiene la responsabilidad de crear objetos de otra clase si agrega, contiene y usa exhaustivamente objetos de dicha clase. Además posee la información necesaria de inicializar esta última clase, es decir: Asignar a la clase B la responsabilidad de crear una instancia de la clase A si se cumple una o más de los casos siguientes:

- B agrega objetos de A
- B contiene objetos de A
- B registra instancias de objetos de A
- B utiliza más estrechamente objetos de A
- B tiene los datos de inicialización que se pasara a un objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A)

Si se asigna bien la responsabilidad de creación, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización (32).La **Figura 7** muestra un ejemplo del empleo de éste patrón en el sistema.

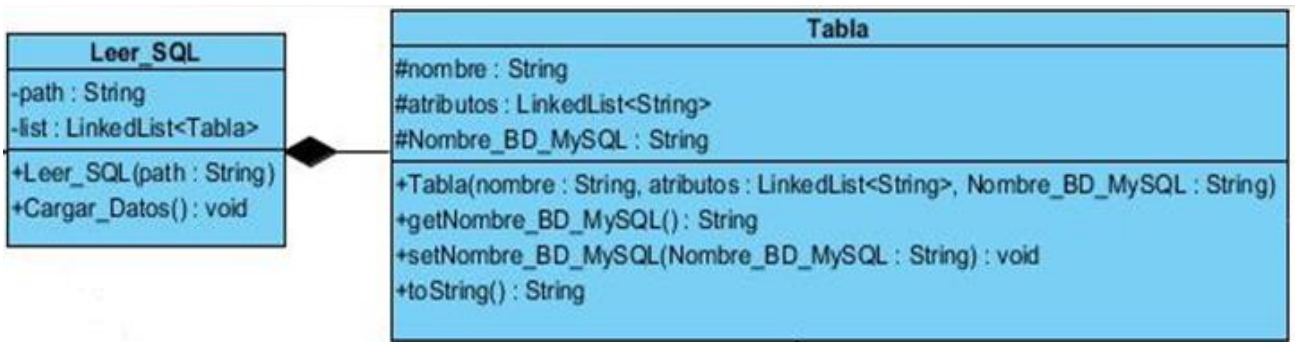


Figura 7 Ejemplo del patrón Creador

Alta cohesión: cohesión funcional dentro de una clase, es una medida que indica cuan relacionadas están las responsabilidades de una clase. Es un patrón evaluativo: entre más alta cohesión más fácil de entender, de cambiar, de reutilizar. No puede ser considerado aisladamente. Una clase con baja cohesión hace muchas cosas no relacionadas, o hace demasiado trabajo. Tales clases no son convenientes, adolecen de los siguientes problemas:

- Difíciles de entender
- Difíciles de reutilizar

- Difíciles de mantener
- Delicadas, constantemente afectadas por los cambios.

Como regla empírica, una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidad altamente relacionada, y no realiza mucho trabajo. Colabora con otros objetos para compartir el esfuerzo si la tarea es extensa (32).La **Figura 8** muestra un ejemplo del empleo de éste patrón en el sistema.

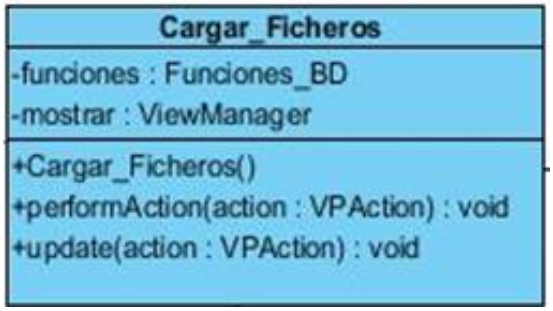


Figura 8 Ejemplo del patrón Alta cohesión

Bajo Acoplamiento: el acoplamiento es la medida de cuánto una clase está conectada (tiene conocimiento) de otras clases. Es un patrón evaluativo: un bajo acoplamiento permite que el diseño de clases sea más independiente. Reduce el impacto de los cambios y aumenta la reutilización. No puede ser considerado aisladamente. Puede no ser importante si la reutilización no es un objetivo (32). En la **Figura 6** anteriormente presentada se evidencia el empleo del patrón en el sistema.

2.5.2 Patrones de diseño GOF[7]

Los Patrones de Diseño GOF en el campo del Diseño Orientado a Objetos son los más conocidos y usados en la actualidad. Se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento; y respecto a su ámbito en clases y objetos.

Respecto a su propósito

- Creacionales: Resuelven problemas relativos a la creación de objetos.
- Estructurales: Resuelven problemas relativos a la composición de objetos.
- Comportamiento: Resuelven problemas relativos a la interacción entre objetos.

[7]Acrónimo de Gang Of Four (Pandilla de los cuatro)

Respecto a su ámbito

- Clases: Relaciones estáticas entre clases.
- Objetos: Relaciones dinámicas entre objetos.(25)

Factory Method o Método de fabricación: es de tipo creación a nivel de clases. Este consiste en definir una interfaz para crear un objeto, permitiendo a las subclases decidir de qué clase instanciar y que una clase delegue en sus subclases la creación de objetos. Se debe utilizar cuando una clase no puede adelantar las clases de objetos que debe crear, cuando una clase pretende que sus subclases especifiquen los objetos que ella crea, cuando una clase delega su responsabilidad hacia otra clase, entre varias subclases auxiliares y se quiere tener localizada a la subclase delegada (33).La **Figura 9** muestra un ejemplo del empleo de éste patrón en el sistema.

```
private static Funciones_BD intanciar = null;
private DiagramManager administrador_diagrama = ApplicationManager.instance().getDiagramManager();
private ViewManager mostrar = ApplicationManager.instance().getViewManager();
```

Figura 9 Ejemplo del patrón Método de fabricación

Iterator o Iterador: es de tipo comportamiento a nivel de objetos. A través de su utilización es posible acceder de forma secuencial a cada uno de los elementos de un objeto agregado sin exponer su representación interna. Además, permite realizar recorridos sobre objetos compuesto sin dependientemente de la implementación de estos. Su utilización tributa a incremento de la flexibilidad porque es posible utilizar nuevas formas de recorrer una estructura con solo modificar el Iterador en uso, cambiarlo por otro o definir uno nuevo. También se facilitan el paralelismo y la concurrencia, pues es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente. Este patrón debe aplicarse cuando se necesite acceder a los elementos de un objeto agregado sin mostrar su representación interna, cuando se requiera hacer recorridos múltiples en objetos agregados y cuando se quiera proporcionar una interfaz uniforme para recorrer diferentes estructuras de agregación (33).La **Figura 10** muestra un ejemplo del empleo de éste patrón en el sistema.

```
public LinkedList<IAttribute> obtenerAtributos(IClass clase) {
    LinkedList<IAttribute> listaatributos = new LinkedList<IAttribute>();
    Iterator atributos = clase.attributeIterator();

    while (atributos.hasNext()) {
        IAttribute atributo = (IAttribute) atributos.next();
        listaatributos.add((IAttribute) atributo);
    }
```

Figura 10 Ejemplo del método Iterador

Conclusiones parciales

Teniendo en cuenta las necesidades que motivan la confección de la extensión, se elaboró una propuesta de solución al problema identificado. Esta solución es guiada por las fases de planificación y diseño planteadas por la metodología utilizada. Se creó el modelo de dominio permitiendo una mejor compresión dentro del contexto del sistema. La identificación de los requisitos funcionales y no funcionales permitió definir las características y las condiciones que debe proveer el plugin. Se definieron un total 6 historias de usuarios en las cuales quedaron agrupados los requisitos funcionales. Se generó además el plan de iteraciones, para de esta forma especificar en la iteración que se desarrolla cada historia de usuario. Se describen un total de 14 tarjetas CRC como parte del diseño en la metodológica XP. Finalmente se describen los patrones de diseño GRASP y GOF, empleados en el proceso de desarrollo de la extensión, facilitando así, la asignación adecuada de las responsabilidades a cada una de las clases en el sistema.

CAPÍTULO 3. IMPLEMENTACIÓNYPRUEBA DE LA EXTENSIÓN PARALA ELABORACIÓN DEL MAPA LÓGICO DE DATOS

En el presente capítulo se realiza la descripción del código fuente de las principales funcionalidades codificadas para la extensión, sirviendo como apoyo las tareas planificadas para el progreso satisfactorio de cada una de las historias de usuarios. La representación de este código está regida por un grupo de estándares de codificación ya creados. Se obtienen resultados del diseño implementando el plugin y se avala la calidad del software mediante la realización de pruebas de software.

3.1 Implementación

En vista a un progreso satisfactorio y un carácter evolutivo de la implementación, el desarrollador concibe un conjunto de tareas de ingeniería asociadas a cada historia de usuario. Estas tareas de desarrollo son definidas en la fase de planificación, las mismas representan fragmentos del grupo de acciones que se realizan para lograr la correcta culminación de cada una de las historias de usuarios en el ámbito de codificación. Las tareas ayudan a organizar la implementación y son solamente usadas por los programadores, pues pueden estar descritas en un lenguaje técnico y no serían necesariamente entendibles por el cliente.

Acorde a la planificación realizada en el capítulo anterior, se llevaron a cabo cinco iteraciones, obteniéndose como finalidad un producto funcional con las características deseadas. A continuación se exponen ejemplos de algunas tareas de ingeniería planificadas, para las historias de usuario uno y tres, correspondientes a las iteraciones número uno y número tres respectivamente.

Iteración 1

En esta iteración se desarrolló la historia de usuario número uno, para la cual se concibieron 12 tareas de desarrollo. La **Tabla 14** muestra la tarea número cuatro, la cual planifica la implementación de la funcionalidad que permite visualizar las tablas y atributos correspondientes al fichero xml que se desea cargar.

Tabla 14 Tarea de ingeniería 4 ¨Diseño de la interfaz para listado de nodos xml¨

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:**1
Nombre Tarea: Diseño de la interfaz para listado de nodos xml	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.2
Fecha Inicio: 20/01/2013	**Fecha Fin:** 22/01/2013
Programador Responsable: Eudel Campuzano-Liena Calzado	
Descripción: Se implementa el diseño de la interfaz que permite visualizar las tablas y atributos correspondientes al fichero	

La figura representa como se visualiza esta tarea ya implementada, en la aplicación.

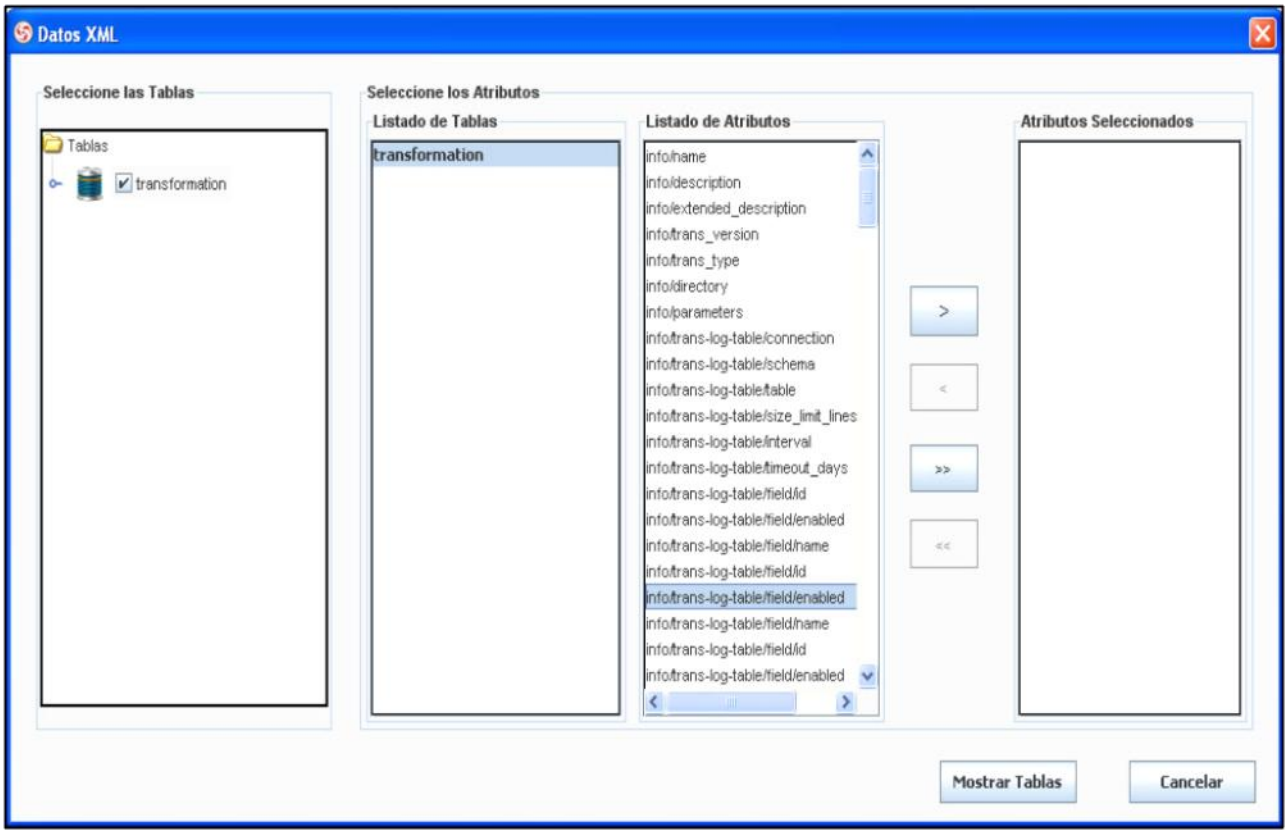


Figura 11 Interfaz que visualiza las tablas y atributos correspondientes al fichero xml

Iteración 3

En esta iteración se desarrolló la historia de usuario número tres, para la cual se concibieron 6 tareas de desarrollo. Las siguientes tablas muestran dos de las tareas definidas, las cuales planifican la implementación de las funcionalidades que permiten visualizar las reglas de transformación de los datos y la relación de los datos fuentes y destino.

Tabla 15 Tarea de ingeniería 5 ¨Diseño de la interfaz para la relación de los datos fuentes y destino¨

Tarea de Ingeniería	
Número Tarea: 5	**Número Historia de Usuario:** 3
Nombre Tarea: Diseño de la interfaz para la relación de los datos fuentes y destino	
Tipo de Tarea : Desarrollo	**Puntos Estimados**: 0.5
Fecha Inicio: 25/04/2013	**Fecha Fin:** 30/04/2013
Programador Responsable: Eudel Campuzano-Liena Calzado	
Descripción: Se implementa el diseño de la interfaz para visualizar la relación de los datos fuentes y destino	

Tabla 16 Tarea de ingeniería 6 ¨Diseño de la interfaz para las reglas de transformación ¨

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario:** 3
Nombre Tarea: Diseño de la interfaz para las reglas de transformación	
Tipo de Tarea : Desarrollo	**Puntos Estimados**: 0.5
Fecha Inicio: 30/04/2013	**Fecha Fin:** 05/05/2013
Programador Responsable: Eudel Campuzano-Liena Calzado	
Descripción: Se implementa el diseño de la interfaz para visualizar las reglas de transformación de los datos	

Las siguientes figuras representan como se visualizan estas tareas ya implementadas, en la aplicación.

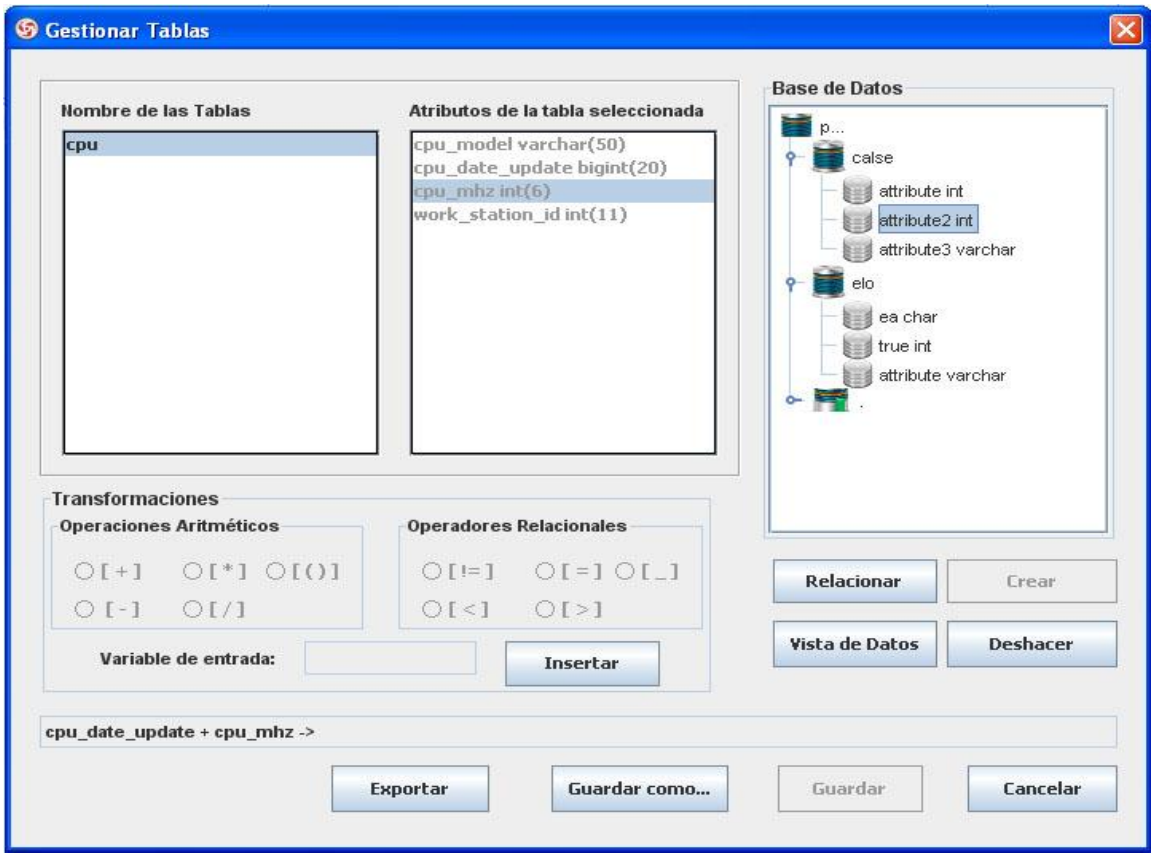


Figura 12 Interfaz que visualiza la relación de los datos fuentes y destino

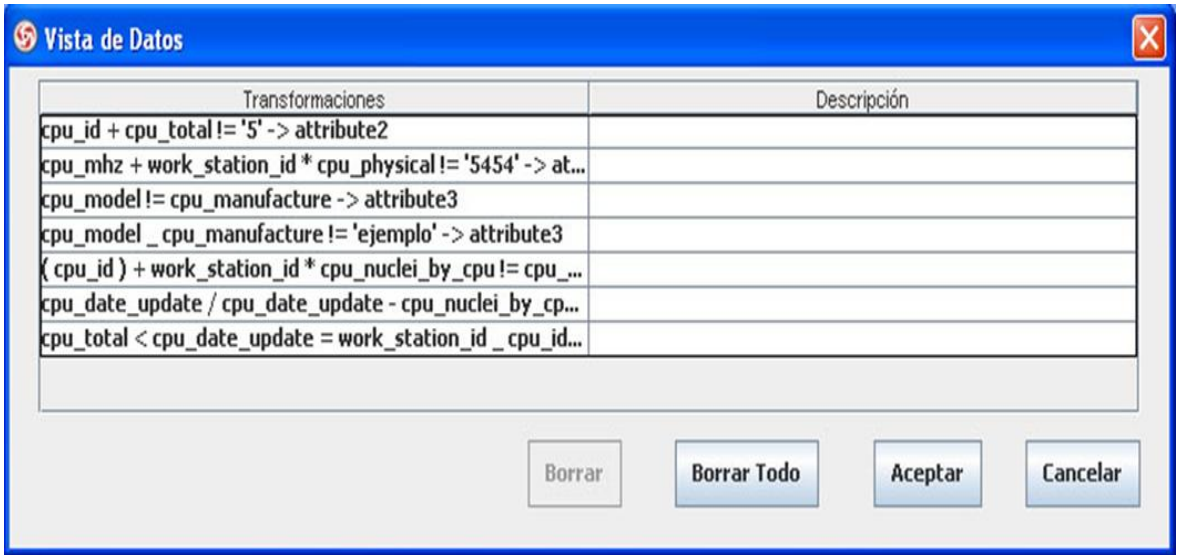


Figura 13 Interfaz que visualizar las reglas de transformación de los datos

Es importante tener en cuenta que cuando se realiza la programación de cada una de las funcionalidades, es esencial mantener una armonía o representación organizada del código, para lograr la factibilidad del manejo de las estructuras deseadas. Por este motivo es necesario regir la implementación por estándares de codificación.

3.2 Estándar de codificación

Un estándar de codificación comprende todos los aspectos de la generación del código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. El mantenimiento del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y el mantenimiento son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es, establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. (34)

Cuando se adopta un estándar de codificación en un proyecto, se establecen un conjunto de convenciones que rigen el proceso de generación del código. Algunas de estas convenciones se pueden ver a continuación:

Organización de los ficheros

Serán evitados los ficheros de gran tamaño que contengan más de 1000 líneas de código, pues en ocasiones el tamaño excesivo provoca que la clase no encapsule un comportamiento claramente definido, albergando una gran cantidad de métodos que realizan tareas funcional o conceptualmente heterogéneas.

Tamaño y organización de las líneas de código

La longitud de línea no debe superar los 80 caracteres. En caso de que una expresión ocupe más de una línea, esta se podrá romper o dividir tras una coma o antes de un operador y la nueva línea debe estar alineada con el inicio de la expresión al mismo nivel que la línea anterior.

Comentarios

Existen dos tipos de comentario, los de implementación y los de documentación, estos últimos existen solo en Java y se encuentran limitados por /**…*/.

Espacios en blanco

Las líneas y espacios en blanco mejoran la legibilidad del código permitiendo identificar las secciones de código relacionadas lógicamente. Se utilizarán espacios en blanco en los siguientes casos: entre una palabra clave y un paréntesis. Esto permite que se distingan las llamadas a métodos de las palabras clave.

Declaraciones de clases

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

- No debe existir ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- Los nombres de las clases comienzan siempre por mayúsculas y si son nombres compuestos se separan por un guion bajo "_".
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"
- Los métodos se separan con una línea en blanco.

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}

    ...
}
```

Sentencia if, if-else, if else-if else

La clase de sentencias if-else debe tener la siguiente forma y deben usar siempre llaves {}.

```
if (condicion) {
    sentencias;
}

if (condicion) {
    sentencias;
} else {
    sentencias;
}

if (condicion) {
    sentencia;
} else if (condicion) {
    sentencia;
} else{
    sentencia;
}
```

Sentencias for

Una sentencia for debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

Sentencias while

Una sentencia while debe tener la siguiente forma:

```
while (condicion) {
    sentencias;
}
```

Variables

Todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres subguión "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje.

Métodos

Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúsculas.

3.3 Descripción del proceso de relación entre los datos fuente y destino en "Visual Paradigm for UML"

Luego de agregar la extensión en la herramienta "Visual Paradigm for UML", el proceso de relacionar los datos fuentes y destino, se realiza a partir de la interpretación de los datos del origen (fuente) y el destino (almacén). La fuente se interpreta mediante los script definidos o la conexión a un gestor de base datos PostgreSQL o MySQL, y en el destino por medio de la obtención de los componentes presentes en los diagramas de clases activos una vez que el usuario se halla conectado a la base de datos mediante el VP.

Para la interpretación de los script definidos se realizan un conjunto de acciones. En este caso se explicará script .sql, el mismo, representa un proceso sencillo, para el desarrollo de este se emplea la clase Scanner brindada por el lenguaje. Esta clase permite ir moviéndose por el fichero, obteniendo de este los nombres de las clases, sus atributos y tipo de dato, guardándolo en una lista de objeto de tipo Tabla, la cual está compuesta por un nombre, una lista de atributos y el nombre de la base datos en el caso de que los datos fuente sean generados mediante la conexión al gestor.

Para establecer la conexión a los gestores MySQL y PostgreSQL primeramente se emplean las librerías mysql-connector-java-5.1.18 y postgresql-9.1-901.jdbc4 respectivamente, luego se verifica la conexión con los parámetros necesarios. Seguidamente mediante consultas al catálogo de los gestores, se obtiene los nombres de todas sus tablas, sus columnas con su tipo de dato y el nombre de la base de datos a la cual se ha conectado, guardándolo en el Objeto de tipo Tabla ya definido.

Para obtener los datos en el destino se crea una instancia de la clase DiagramManager mediante la interfaz ApplicationManager, la cual permite obtener el diagrama activo mediante el método getActiveDiagram(), obteniendo con este el nombre del diagrama activo, el cual se toma como el nombre de la base datos destino, además se capturan los elementos IClass , y una vez obtenidos se capturan los elementos IAttribute asociados a cada elemento IClass. El elemento IClass brinda como información el nombre de la tabla y el tipo de estereotipo, y el elemento de IAttribute el nombre de los atributos. De esta forma se obtiene la lista de Tabla_BD la cual está compuesta por los nombres de la tabla que tiene la base de datos, la lista de atributos, el nombre de la base de datos y las dimensiones lentamente cambiantes, esta última información introducida por el usuario posteriormente.

Una vez obtenida la lista de tablas de la fuente y del destino el usuario debe seleccionar de manera visual las tablas y los atributos que necesita para establecer las relaciones entre los datos fuentes y destino. Para esto se debe instanciar el visual correspondiente al fichero o base datos seleccionada en la fuente, la cual es visualizada mediante la clase interfaz ApplicationManager brindada por la librería OpenApi. Luego seleccionada las clases y sus respectivos atributos, se debe instanciar la clase visual Gestionar_Tablas, encargada de llevar a cabo el proceso de transformación de los atributos fuentes y destino.

Cuando se realiza la implementación de la solución, el único instrumento adecuado para determinar la calidad y efectividad del código y el sistema, es el proceso de pruebas.

3.4 Pruebas de software

La prueba del software es un elemento crítico para garantizar la calidad del mismo, por lo que el objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna y las funciones externas del software, las cuales consisten en la ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si descubre un error no detectado hasta entonces. (35)
La etapa de pruebas implica acciones como:

- Verificar la interacción de los componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

XP es considerada una metodología “Test-drivenprogramming” (guiada por pruebas). En esta se planifican varios ciclos de iteraciones que a su vez contienen una o muchas historias de usuario, a diferencia de las metodologías tradicionales en las que la fase de prueba queda recluida por definición al culminar las actividades de desarrollo. Contrario a este sentido XP propone planificar y crear las pruebas antes de implementar cualquier solución del proyecto. La definición de estas pruebas al comienzo, condiciona o “dirige” el desarrollo del proyecto ya que al plantearse las pruebas con anterioridad el desarrollador solo deberá preocuparse por realizar el esfuerzo mínimo necesario para pasar las pruebas anteriormente planteadas, agilizando así el proceso de obtención de la solución. Este principio tiene ventajas y desventajas ampliamente discutidas por los entendidos en la materia sin embargo la definición de “Ágil” de la metodología respalda el acto. La metodología XP plantea además una fase prueba para cada iteración. Al concluir cada ciclo se realizarán las pruebas, analizando cada historia de usuario, así como en cada ciclo consiguiente, con el objetivo de verificar que las iteraciones posteriores no afecten a las anteriores. Las pruebas que hayan sido fallidas en el ciclo anterior se analizan para evaluar su corrección y prevenir su futura ocurrencia.

Para determinar el grado de cumplimiento de las especificaciones iniciales de la extensión, se realizarán pruebas a la aplicación, en las cuales, el sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas. Los resultados obtenidos se observarán y registrarán con el fin de realizar una evaluación o dar solución a algún resultado no deseado. Las pruebas no pueden asegurar ausencia de defectos; sólo pueden demostrar que existen defectos en el software. Las pruebas que se realizarán son:

Pruebas unitarias: las pruebas unitarias durante el desarrollo de software incrementan exponencialmente la calidad del código producido y verifica todos los flujos de control. Los programadores desarrollarán las pruebas antes de escribir el código que logrará que éstas pasen, así escribirán las pruebas unitarias a medida que piensen el código. Para cada uno de los métodos del código que pudiese fallar, deberá ser escrita la correspondiente prueba unitaria. (36)

Pruebas de sistema: permiten asegurar la apropiada navegación dentro del sistema, el ingreso de datos, procesamiento y recuperación. Estas pruebas se enfocan principalmente en los requisitos funcionales del software. Posibilitan obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignorará la estructura de control, concentrándose en los requisitos y ejercitándolos. (37)

Estas pruebas permiten encontrar:

- Funciones incorrecta o ausente.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación. (37)

Las pruebas de sistema se apoyan en el artefacto Caso de prueba para su elaboración. El mismo ayuda a comprobar el correcto funcionamiento de los componentes del sistema de información, analizando un conjunto de entradas y condiciones de ejecución verificando que el resultado es el esperado. Se consideran exclusivamente las entradas y salidas del sistema sin preocuparse por la estructura interna del mismo. (37)

Pruebas de aceptación: prueba final antes del despliegue del sistema. Es ejecutada antes de que la aplicación sea instalada dentro de un ambiente de producción. La realiza el cliente permitiéndole determinar la aceptación o rechazo del sistema desarrollado. Si el cliente está de acuerdo con la aplicación, se emite una carta de aceptación. (38). Estas pruebas se apoyan en los Casos de prueba por escenarios o matriz de datos.

Luego de escoger las pruebas que se le realizarán a la aplicación, se efectuarán las mismas en dependencia del ámbito al que están asignadas. Las pruebas unitarias serán ejecutadas por los desarrolladores al concluir cada iteración. Las pruebas de sistema la realizarán los tutores, a la interfaz del sistema, apoyándose en los casos de prueba como guía en el proceso, y las pruebas de aceptación serán hechas por los asesores de calidad del DAD (clientes). Estas pruebas arrojaran los resultados que permitirán conocer el estado de la aplicación.

3.5 Resultados de las pruebas hechas al sistema

En la ejecución de las pruebas, correspondientes a los respectivos contextos a los que corresponden cada una de ellas, se detectaron un grupo de No Conformidades (NC), las cuales se clasificaron en alta, media o baja, en dependencia del grado de dificultad que muestran y lo que representan dentro del sistema. A continuación se exponen los resultados de las mismas:

3.5.1 Pruebas unitarias

Mediante las pruebas unitarias realizadas al código por los desarrolladores, se detectaron un total de 14 NC, la clasificación y distribución de estas se muestra a continuación en la **Figura 14**.

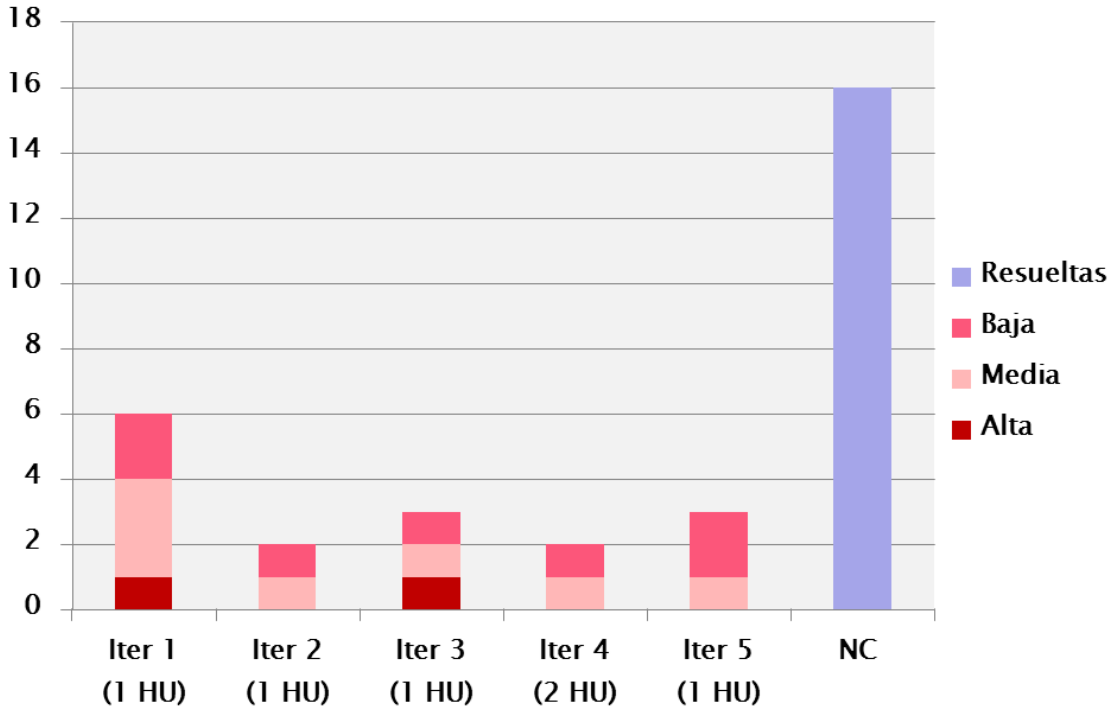


Figura 14 Distribución de las NC por iteración

3.5.2 Pruebas de sistema

Las pruebas de sistema realizadas por los tutores a la interfaz de la aplicación, arrojo un total de 7 NC. A continuación se exponen tres de los casos de prueba correspondientes a la historia de usuario número uno ¨Cargar los ficheros desde la fuente¨ y la **Tabla 20** evidencia el número de NC clasificadas en alta, media o baja según su dificultad.

Tabla 17 Prueba 1 de la HU "Cargar los ficheros de la fuente"

Caso de prueba de aceptación	
Código: HU1_p1	**Historia de Usuario:** 1
Nombre: Cargar ficheros con extensión xml	
Nombre de la persona que realiza la prueba: Ramón Ernesto Stevenson Borrell	
Descripción de la Prueba: Probar que el sistema permite cargar ficheros con extensión xml mostrando las tablas con sus atributos	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección del mapa lógico de datos el usuario debe seleccionar la opción que permite cargar el fichero y localizarlo en la dirección deseada	
Resultado esperado: Que la aplicación permita cargar el fichero que está en una dirección específica, mostrando las tablas con todos sus atributos	
Evaluación de la prueba: Satisfactoria	

Tabla 18 Prueba 2 de la HU "Cargar los ficheros de la fuente"

Caso de prueba de aceptación	
Código: HU1_p2	**Historia de Usuario:** 1
Nombre: Cargar el script .sql generado por los gestores de base de datos PostgreSQL y MySQL	
Nombre de la persona que realiza la prueba: Ramón Ernesto Stevenson Borrell	
Descripción de la Prueba: Probar que el sistema permite cargar el script .sql generado por los gestores de base de datos PostgreSQL o MySQL mostrando las tablas con sus atributos.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección del mapa lógico de datos el usuario debe seleccionar la opción que permite cargar el fichero y localizarlo en la dirección deseada.	

Resultado esperado: Que la aplicación permita cargar el fichero que está en una dirección específica, mostrando las tablas con todos sus atributos.
Evaluación de la prueba:

Tabla 19 Prueba 3 de la HU "Cargar los ficheros de la fuente"

Caso de prueba de aceptación	
Código: HU1_p3	**Historia de Usuario:** 1
Nombre: Cargar ficheros con extensión xls	
Nombre de la persona que realiza la prueba: Yanisbel González Hernández	
Descripción de la Prueba: Probar que el sistema permite cargar ficheros con extensión xls mostrando las tablas con sus atributos.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección del mapa lógico de datos el usuario debe seleccionar la opción que permite cargar el fichero y localizarlo en la dirección deseada.	
Resultado esperado: Que la aplicación permita cargar el fichero que está en una dirección específica, mostrando las tablas con todos sus atributos.	
Evaluación de la prueba:	

Tabla 20 NC detectadas por la pruebas de sistema

No. NC	Alta	Media	Baja	Resueltas
7	**-**	**2**	**5**	**7**

3.5.3 Pruebas de aceptación

Las pruebas de aceptación realizadas por los asesores de calidad del DAD a la aplicación, arrojo un total de 5 NC, las cuales referían a errores de concordancia y uso indebido o excesivo de mayúsculas (Ej: Existen atributos sin seleccionar del Listado de Tablas, Ej: Guardar Proyecto).). A continuación se muestran las variables que representan valores de entrada de datos y la matriz de datos, para comprobar que el plugin funcione de forma correcta, donde:

- V: indica válido, I: indica inválido, NA: que no es necesario proporcionar un valor del dato, ya que es irrelevante.
- En la siguiente tabla se muestran las variables que representan valores de entrada de datos para el caso de prueba basado en la HU 1.

Tabla 21 Descripción de las variables de la matriz de datos basada en la HU 1

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Base datos destino	campo de texto	No	Muestra el nombre de la solución que se va a crear.

Tabla 22 Matriz de datos basada en la HU 1

Escenario	Descripción	Base datos destino	Respuesta del sistema	Flujo central
EC 2.1 Cargar los ficheros de la fuente de datos correctamente.	El usuario carga la base de datos destino y selecciona el fichero que desea cargar.	V	El sistema muestra una interfaz con todas las tablas y atributos de la base de datos.	1-El usuario carga la base de datos destino mediante el Visual Paradigm o carga un diagrama de clases de la base de datos destino. 2-El usuario da clic en el botón Ext Vpml de la barra de tareas del Visual Paradigm for UML, seguidamente en el botón Cargar fuente y después en Cargar ficheros. 3-El usuario especifica el fichero que desea cargar.

				4- El usuario da clic en el botón Abrir para cargar el fichero.
EC 2.2 Cargar los ficheros de la fuente de datos incorrecta mente.	El usuario no carga la base de datos destino.	I	El sistema muestra un mensaje de error ("Debe de cargar alguna base de datos en el Visual Paradigm").	1-El usuario no carga la base de datos destino.

En respuesta a las No Conformidades detectadas, se trabajó en la identificación de la dificultad y su ajuste o solución, según el caso correspondiente. Algunos de los inconvenientes señalados se debían a temas de programación, navegación, gramática o estandarización de los campos, los que se resolvieron de manera satisfactoria. Las NC detectadas por el cliente fueron solucionadas en una segunda iteración emitiéndose así por parte de este la carta de aceptación, lo que constata que el producto está listo para su utilización.

Conclusiones parciales

En el proceso de construcción de la solución se le dio cumplimiento a un conjunto de tareas de la implementación en las 5 iteraciones realizadas, dándole cumplimiento a los requisitos funcionales que fueron agrupados en las 6 historias de usuario definidas. Se le realizaron las pruebas unitarias, de sistema y de aceptación al producto, lográndose validar la calidad de las funcionalidades del mismo, calificando la solución obtenida, como satisfactoria.

CONCLUSIONES

Luego de finalizada la investigación e implementada la solución, se arriba a las siguientes conclusiones:

- En concordancia con el estudio realizado y por ser las fuentes utilizadas en el departamento Almacenes de datos, se determinó que, los orígenes de datos que van a ser manejados en el diseño y confección del Mapa Lógico de Datos son las Hojas de cálculo, los ficheros XML y la conexión a las fuentes de datos mediante los Gestores de Base de Datos PostgreSQL y MySQL, además los ficheros SQL generados por estos gestores.
- La metodología que se adecua a las características del proyecto y al equipo de desarrollo es XP por lo que se escoge como metodología de desarrollo de software y se emplea como herramienta CASE Visual Paradigm e IDE NetBeans 7.1.
- Para definir las características y condiciones que debe proveer la solución se identificaron 6 historias de usuarios, las cuales describen los requisitos funcionales de la extensión, además se realizaron 14 tarjetas CRC que permitieron una correcta planificación en aras de satisfacer las necesidades del departamento Almacenes de datos.
- En el proceso de construcción de la solución se le dio cumplimiento a un conjunto tareas de implementación que permitieron el correcto desarrollo de la extensión, además se le realizaron las pruebas unitarias, de sistema y de aceptación al producto, lográndose validar la calidad de las funcionalidades del mismo, calificando la solución obtenida como satisfactoria.

RECOMENDACIONES

- Utilizar el contenido de la investigación como base de referencia para el desarrollo de futuras extensiones para la herramientas CASE “Visual Paradigm for UML“.
- En el desarrollo de futuras versiones agregar un requisito que permita interpretar los diagramas de clases como fuente de datos.

ANEXOS

Anexo 1 Ejemplo de la implementación del archivo Plugin.xml

```
<plugin
id="sample.plugin"
name="Sample Plugin"
description="Sample Plugin"
provider="Visual Paradigm"
class="sample.plugin.SamplePlugin">
        <runtime>
                <library path="lib/sampleplugin.jar" relativePath="true"/>
                <library path="lib/AbsoluteLayout.jar" relativePath= "true"></library>
        </runtime>
        <actionSets>
                <actionSet id="sample.plugin.actions.ActionSet1">
                        <toolbar
                                id="sample.plugin.actions.Toolbar1"
                                orientation="north"
                                index="last"/>
                        <menu
                                id="sample.plugin.actions.Menu1"
                                label="Sample Menu 1"
                                mnemonic="M"
                                menuPath="Tools/Report"/>
                        <action
                                id="sample.plugin.actions.Action1"
                                actionType="generalAction"
                                label="Sample Action 1"
                                tooltip="Sample Action 1"
                                icon="icons/red.png"
                                style="normal"
                                menuPath="Tools/Report"
                                toolbarPath="sample.plugin.actions.Toolbar1/#"
                                actionController class="sample.plugin.actions.ActionController"/>
                        </action>
```

```
                    <separator
                        id="sample.plugin.actions.Separator1"
                        menuPath="Tools/sample.plugin.actions.Action1"
                        toolbarPath="sample.plugin.actions.Toolbar1/sample.plugin.action.Action1"/>
                    </actionSet>
        <contextSensitiveActionSet id="sample.plugin.actions.ActionSet2">
            <contextTypes all="false">
            <include type="Class"/>
            <exclude type="Package"/>
            </contextTypes>
                <action
                    id="sample.plugin.actions.ContextAction1"
                    label="Sample Action [1]"
                    icon="icons/blue.png"
                    style="toggle"
                    menuPath="OpenSpecification">
                    actionController class="sample.plugin.actions.ContextActionController"/>
                </action>
            </contextSensitiveActionSet>
        </actionSets>
</plugin>
```

Anexo 2 Ejemplo de la implementación de las clases VPActionController y ContextActionController

```
public class ActionController implements com.vp.plugin.action.VPActionController {
ViewManager view = ApplicationManager.instance ().getViewManager();
 public void performAction(VPAction action) {
   view.showMessage("Mi primer plugin");
 }
 public void update(VPAction action) {
   throw new UnsupportedOperationException("Not supported yet.");
 }
}
public class ContexActionController implements com.vp.plugin.action.VPContexActionController {
 ViewManager view = ApplicationManager.instance().getViewManager();
public void performAction(VPAction action, VPContext vpc, ActionEvent ae) {
      view.showMessage("Mi primer plugin");
    }
public void update(VPAction action, VPContext vpc) {
   throw new UnsupportedOperationException("Not supported yet.");
    }
 }
```

GLOSARIO DE TÉRMINOS

CASE: de sus siglas en inglés Cumputer Aided Software Engineering es un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de Software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software.

ETL: proceso encargado realizar la compleja actividad de extraer datos de diferentes fuentes, para luego integrarlos, filtrarlos y depurarlos.

Integración de datos: proceso de unificación de los datos provenientes de múltiples fuentes.

Mapa Lógico de Datos: artefacto que describe el flujo de los datos desde el origen (fuentes de datos) hasta el destino (almacén de datos).

Refactorización: (del inglés Refactoring) es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

UML: De sus siglas en inglés Unified Modelling Language, Lenguaje Unificado de Modelado es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar artefactos de un sistema de software, permite realizar modelos para la guía del proceso de desarrollo del software.

REFERENCIAS BIBLIOGRÁFICAS

1. **AGUILAR, Joyanes.** *La gestión del Conocimiento en la Comunicación: Un enfoque Tecnológico y de Gestión de Contenidos.* Madrid : s.n., 2002. ISBN 978-958-719-053-3.

2. **PABLOS, Carmen de.** *Informática y comunicaciones en la empresa.* Madrid : 1ra ed, 2004. ISBN 84-7356-375-1.

3. **KIMBALL Ralph, Joe Caserta.** *The Data Warehouse ETL Toolkit.* Indianapolis : 1ra ed, 2004. ISBN 0-764-56757-8.

4. **KIMBALL, Ralph.** *The Data Warehouse Toolkit.* Estados Unidos : 1ra ed, 1996. ISBN 978-0-470-56310-6.

5. **WHITE, Colin.** *Data Integration: Using ETL, EAI, and EII Tools to Create an Integrated Enterprise.* Estados Unidos : 2da ed, 2006. ISBN 0-471-08130-2-8.

6. **TECHNOLOGY, Denodo.** Denodo Technology. [En línea] [Citado el: 12 de Enero de 2012.] http://www.denodo.com.

7. **MORGENTHAL, JP.** *Enterprise Information Integration: A Pragmatic Approach.* s.l. : 1ra ed, 2006. ISBN 88-85280-62-5.

8. **ESPINOSA, Itziar Angoiti.** *Data warehouse para la gestión de lista de espera sanitaria.* 2011.

9. **ECURED.** *Ecured: Enciclopedia cubana*. [En línea] [Citado el:15 de noviembre de 2012.] http://www.ecured.cu/index.php/Sistema_Gestor_de_Base_de_Datos.

10. **DEFINICION**. [En línea] [Citado el: 10 de Febrero de 2013.] http://definicion.de/categoria/tecnologia/Definición%de%XML.

11. **PEREZ, Miguel Martínez.** Hojas de Cálculo. España : 4ta ed, 2010. ISBN 978-84-8454-970-3.

12. **VISUAL, PARADIGM**. Visual Paradigm. [En línea] [Citado el: 15 de Diciembre de 2012.] http://www.visual-paradigm.com/.

13. **MENENDEZ, Rafael**. Informatica Aplicada a la GestionPublica. IAGP, Metodologías de desarrollo de software. Universidad de Murcia : s.n., 2011.

14. **INFORMATICOS, D. D. L. Y. S.** *Rational Unified Process (Rup).* Escuela Técnica Superior de Ingienería Informática, Universidad de Sevilla : s.n., 2006. ISBN 1-59140-851-2.

15. **TOROSSI, A. U. S. G.** *El Proceso Unificado de Desarrollo de Software.* 2003.

16. **CANOS, P. L. Y. C. P José H.** *Métodologías Ágiles en el Desarrollo de Software.* Universidad Politécnica de Valencia : s.n., 2008. ISBN 978-987-663-001-6.

17. **SANCHEZ, María A. Mendoza.** *Metodologías De Desarrollo De Software.* s.l. : 1ra ed, 2004. ISBN 84-8468-043-6.

18. **ESCRIBANO, Gerardo Fernández.** *Introducción a Extreme Programming.* s.l. : 1ra ed, 2002. ISBN: 0-201-. 71091-9.

19. **WIGAHLUK.** *Entre la XP y el RUP?* [En línea] [Citado el: 20 de Diciembre de 2012.] http://wigahluk.wordpress.com/2007/06/26/entre-la-xp-y-el-rup/.

20. **FLORES, M. M. D.** *RUP vs XP.* s.l. : 2DA ED, 2006. ISBN 0-201-73039-1.

21. **HERNANDEZ, M. C.** *Historia de los lenguajes de programación.* s.l. : 4ta ed, 2011. ISBN 978-98001-1611-1.

22. **GAYO, J. E. L.** Lenguaje de XML. [En línea] [Citado el: 19 de Diciembre de 2012.] http://www.topxml.com.

23. **OMG.** *Unified Modeling LanguageTM (OMG UML).* s.l. : 2da ed, 2009. ISBN: 0-201-69961-3.

24. **NETBEANS.** *Netbeans.* [En línea] [Citado el: 25 de Enero de 2013.] http://www.netbeans.org.

25**. ORALLO, E. H.** El Lenguaje Unificado de Modelado (UML). [En línea] 2003. [Citado el: 10 de Diciembre de 2012.] http://es.scribd.com/doc/54817695/Act-a-Uml.

26**. YANG, Xiaoyu.** *Visual Paradigm Suite Extensibility Plugin User´s Guide.* NY : s.n., 2011. ISBN 978-0-85729-4395.

27. **MARTINEZ, Juan Francisco Javier.** *Guía de construcción de software en java con patrones de diseño.* s.l. : 1ra ed, 2002. ISBN 84-7829-028-1.

28. **SOMMERVILLE, Ian.** *Software Engineering.* 2006. ISBN 0-321-31379-8.

29. **GIRALDO, O.P.** *Ingeniería de Requisitos.* s.l. : Vol13, 2007. ISBN 84-8021-408-2.

30. **GAMMA ERICH, R. H., Ralph Johnson, John Vlissides.** *Design Pattens-Elements of Reusable.* Estados Unidos : **2da** ed, 2004. ISBN 0-201- 63361-2.

31. **LARMAN, Prentice Hall Craig. *UML y Patrones.* 2003.**

32. **LASATER, Christopher**. *Design Patterns.* Texas : 1ra ed, 2007. ISBN 1-59822-031-4.

33. **RODRIGUEZ, Joaquín Adiego**. Patrones de diseño. [En línea] [Citado el: 3 de Mayo de 2013.] http://www.infor.uva.es/~felix/datos/priii/tr_patrones-2x4.pdf.

34. **TRISTAN,** Fernando Becerra. Estandáres de codificación. [En línea] [Citado el: 10 de Abril de 2013.] http://serk.kualtus.com/codigo.htm.

35. **CETIC.** Tipos Prueba Software. [En línea] [Citado el: 5 de Mayo de 2013.] http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf.

36. **BECK, Kent.** *Extreme Programming Explained: Embrace the Change.* Estados Unidos : 2da ed, 2004. ISBN 201-61641-6.

37. **PRESSMAN, Roger**. *Ingeniería del Software. Un enfoque práctico.* NY : 2da ed, 2001.

38. **HOWER, Rick.** The Complete Guide to software testing. [En línea] [Citado el: 4 de Mayo de 2013.] http://www.softwareqatest.com.

BIBLIOGRAFÍA

1. **AGUILAR, Joyanes.** *La gestión del Conocimiento en la Comunicación: Un enfoque Tecnológico y de Gestión de Contenidos.* Madrid : s.n., 2002. ISBN 978-958-719-053-3.

2. **BECK, Kent.** *Extreme Programming Explained: Embrace the Change.* Estados Unidos : 2da ed, 2004. ISBN 201-61641-6.

3. **CANOS, P. L. Y. C. P José H.** *Métodologías Ágiles en el Desarrollo de Software.* Universidad Politécnica de Valencia : s.n., 2008. ISBN 978-987-663-001-6 .

4. **CETIC.** Tipos Prueba Software. [En línea] [Citado el: 5 de Mayo de 2013.] http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf.

5. **CORTÉS, Oscar Hernando Guzmán**. *Aplicación práctica del diseño de pruebas de software a nivel de programación* [En línea] 2008. [Citado el: 10 de Mayo de 2013.]. http://www.icesi.edu.co/revistas/index.php/sistemas_telematica/article/view/935

6. **CRISTINA.** UML (Informática). [En línea] 2008**.** http://hdl.handle.net/2099.1/11305.

7. **DEFINICION.** [En línea] [Citado el: 10 de Febrero de 2013.] http://definicion.de/categoria/tecnologia/Definición%de%XML.

8. **ECURED.** Ecured: Enciclopedia cubana. [En línea] 15 de noviembre de 2012. http://www.ecured.cu/index.php/Sistema_Gestor_de_Base_de_Datos.

9. **ERRAZURIZ, Alfredo Barros.** EAI. [En línea] 2010. [Citado el: 20 de Enero de 2013.] http://www.technologyevaluation.com/es/software/eai-articulos.html?.

10. **ESCRIBANO, Gerardo Fernández.** *Introducción a Extreme Programming.* s.l. : 1ra ed, 2002. ISBN: 0-201-. 71091-9.

11. **ESPINOSA, Itziar Angoiti.** *Data warehouse para la gestión de lista de espera sanitaria.* 2011.

12. **FLORES, M. M. D.** *RUP vs XP.* s.l. : 2da ed, 2006. ISBN 0-201-73039-1.

13. **GAMMA ERICH, R. H., Ralph Johnson, John Vlissides.** *Design Pattens-Elements of Reusable.* Estados Unidos : 2da ed, 2004. ISBN 0-201- 63361-2.

14. **GARZAS**, **Javier**. Calidad de software [En línea] 2009. [Citado el: 22 de Enero de 2013.] http://www.javiergarzas.com/calidad-software

15. **GAYO, J. E. L.** Lenguaje de XML. [En línea] [Citado el: 19 de Diciembre de 2012.] http://www.topxml.com.

16. **GIRALDO, O.P.** *Ingeniería de Requisitos.* s.l. : Vol13, 2007. ISBN 84-8021-408-2.

17. **GOMEZ, Antonio** .Conceptual Modeling. [En línea] 2011**.** http://upcommons.upc.edu/pfc/.

18. **GOMEZ, Antonio.** CASE tools. [En línea] 2010**.** http://hdl.handle.net/2099.1/11305.

19. **HERNANDEZ, M. C.** *Historia de los lenguajes de programación.* s.l. : 4ta ed, 2011. ISBN 978-98001-1611-1.

20. **HOWER, Rick.** *The Complete Guide to software testing.* [En línea] [Citado el: 4 de Mayo de 2013.] http://www.softwareqatest.com.

21. **INFORMATICOS, D. D. L. Y. S.** *Rational Unified Process (Rup).* Escuela Técnica Superior de Ingienería Informática, Universidad de Sevilla : s.n., 2006. ISBN 1-59140-851-2.

22. **KIMBALL Ralph, Joe Caserta.** *The Data Warehouse ETL Toolkit.* Indianapolis : 1ra ed, 2004. ISBN 0-764-56757-8.

23. **KIMBALL, Ralph.** *The Data Warehouse Toolkit.* Estados Unidos : 1ra ed, 1996. ISBN 978-0-470-56310-6.

24. **LARMAN, Prentice Hall Craig.** *UML y Patrones.* 2003.

25. **LASATER, Christopher.** *Design Patterns.* Texas : 1ra ed, 2007. ISBN 1-59822-031-4.

26. **MARTINEZ, Juan Francisco Javier.** *Guía de construcción de software en java con patrones de diseño.* s.l. : 1ra ed, 2002. ISBN 84-7829-028-1.

27. **MENENDEZ, Rafael.** *Informatica Aplicada a la GestionPublica. IAGP, Metodologías de desarrollo de software.* Universidad de Murcia : s.n., 2011.

28. **MORGENTHAL, JP.** *Enterprise Information Integration: A Pragmatic Approach.* s.l. : 1ra ed, 2006. ISBN 88-85280-62-5.

29. **NETBEANS.** *Netbeans.* [En línea] [Citado el: 25 de Enero de 2013.] http://www.netbeans.org.

30. **OMG.** *Unified Modeling LanguageTM (OMG UML).* s.l. : 2da ed, 2009. ISBN: 0-201-69961-3 .

31. **ORALLO, E. H.** *El Lenguaje Unificado de Modelado (UML).* [En línea] 2003. [Citado el: 10 de Diciembre de 2012.] http://es.scribd.com/doc/54817695/Act-a-Uml.

32. **PABLOS, Carmen de.** *Informática y comunicaciones en la empresa.* Madrid : 1ra ed, 2004. ISBN 84-7356-375-1.

33. **PEREZ, Miguel Martínez.** *Hojas de Cálculo.* España : 4ta ed, 2010. ISBN 978-84-8454-970-3.

34. **PRESSMAN, Roger.** *Ingeniería del Software. Un enfoque práctico.* NY : 2da ed, 2001.

35. **RAMON, Olivé.** UML (Computer science). [En línea] 2010**.** http://hdl.handle.net/2099.1/11305.

36. **RODRIGUEZ, Joaquín Adiego.** Patrones de diseño. [En línea] [Citado el: 3 de Mayo de 2013.] http://www.infor.uva.es/~felix/datos/priii/tr_patrones-2x4.pdf.

37. **SANCHEZ, María A. Mendoza.** *Metodologías De Desarrollo De Software.* s.l. : 1ra ed, 2004. ISBN 84-8468-043-6.

38. **SOMMERVILLE, Ian.** *Software Engineering.* 2006. ISBN 0-321-31379-8.

39. **TECHNOLOGY, Denodo.** Denodo Technology. [En línea] [Citado el: 12 de Enero de 2012.] http://www.denodo.com.

40. **Tendencias actuales EII.** [En línea] 2011. [Citado el: 20 de Enero de 2013.] http://www.gestionypoliticapublica.cide.edu/.../TENDENCIAS%20ACTUALE...?.

41. **TOROSSI, A. U. S. G.** *El Proceso Unificado de Desarrollo de Software.* 2003.

42. **TRISTAN, Fernando Becerra.** Estandáres de codificación. [En línea] [Citado el: 10 de Abril de 2013.] http://serk.kualtus.com/codigo.htm.

43. **VISUAL, PARADIGM.** Visual Paradigm. [En línea] [Citado el: 15 de Diciembre de 2012.] http://www.visual-paradigm.com/.

44. **WHITE, Colin.** *Data Integration: Using ETL, EAI, and EII Tools to Create an Integrated Enterprise.* Estados Unidos : 2da ed, 2006. ISBN 0-471-08130-2-8.

45. **WIGAHLUK.** *Entre la XP y el RUP?* [En línea] [Citado el: 20 de Diciembre de 2012.] http://wigahluk.wordpress.com/2007/06/26/entre-la-xp-y-el-rup/.

46. **YANG, Xiaoyu.** *Visual Paradigm Suite Extensibility Plugin User´s Guide.* NY : s.n., 2011. ISBN 978-0-85729-4395.