**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**FACULTAD 6**

# EXTENSIÓN PARA MODELAR SOLUCIONES DE ALMACENES 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:**

**Ediel Enrique García Amador**

**Alejandro Mariño Pérez**

**Tutores:**

**Ing. José Salvador Bermúdez Rodríguez**

**Ing. Elio Luis Toledo García**

**La Habana, Junio de 2014**

**“Año 56 de la Revolución”**

*El éxito no llega por suerte, es el sacrificio y el esfuerzo de días, meses y años de trabajo.*

*Paublo Coelho*.

## DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firman la presente, a los ____ días del mes de _____ del año _____.

Ediel Enrique García Amador

_____

Firma del Autor

Alejandro Mariño Pérez

_____

Firma del Autor

Ing. José Salvador Bermúdez Rodríguez

_____

Firma del tutor

Ing. Elio Luis Toledo García

_____

Firma del tutor

# DATOS DE CONTACTO

Autores:

Autor: Est. Ediel Enrique García Amador

Universidad de las Ciencias Informáticas

e-mail: eegarcia@estudiantes.uci.cu

Autor: Est. Alejandro Mariño Pérez

Universidad de las Ciencias Informáticas

e-mail: amarinop@estudiantes.uci.cu

Tutores:

Tutor: Ing. José Salvador Bermúdez Rodríguez

Especialidad de graduación: Ingeniería en Ciencias Informáticas

Categoría docente: Instructor

Categoría Científica: -

Años de experiencia en el tema: 4

Años de graduado: 4

Correo Electrónico: jsbermudez@uci.cu

Tutor: Ing. Elio Luis Toledo García

Especialidad de graduación: Ingeniería en Ciencias Informáticas

Categoría docente: -

Categoría Científica: -

Años de experiencia en el tema: 2

Años de graduado: 3

Correo Electrónico: eltoledo@uci.cu

*El éxito no significa nada si no tienes con quien compartirlo, por eso quiero agradecerle a todos los que contribuyeron en el resultado de esta investigación, que sería imposible sin el esfuerzo de un gran número de personas, desde el colaborador más cercano, hasta el que ocasionalmente con una simple frase, me estimuló a continuar adelante. Mencionarlos a todos es difícil. Siéntanse todos, sin excepción, reconocidos en las personas que a continuación relaciono.*

*En primer lugar a Dios por permitirme llagar hasta aquí, por ser mi guía y mi amigo más fiel.*

*A mi mamá por ser la mejor madre del mundo, por ser incondicional, por apoyarme siempre y enseñarme a ser una mejor persona. Por convertirse en madre y padre, amiga sobre todas las cosas, por estar presente en todas las etapas de mi vida. Gracias por enseñarme a ser fuerte y a luchar por alcanzar mis sueños. Por ser la voz que me impulsa y me apoya constantemente en todas las metas que me propongo. Por la educación con la que me has formado, por llevarme a ser alguien en la vida, por inspirar con solamente una mirada, el respeto más grande que pueda sentir por una persona. A ella debo todo lo que soy y mi gran sueño es que nunca me falte.*

*A mi papá Enrique por convertirse en mi verdadero padre, gracias por acogerme a mí y a mis hermanos como tus hijos, por educarnos, por estar siempre presente, por darme una familia paterna, por la educación que has formado en mí, por ser un ejemplo a seguir. Gracias por apoyarme y por guiarme a ser una persona de bien.*

*A mis hermanos Eddy y Ednier por su cariño y dedicación. Gracias por todo el apoyo que siempre me han ofrecido. Por ayudarme a cumplir mis sueños, por estar siempre cerca. El amor incondicional de ustedes me impulsa a ser mejor cada día.*

*A mis hermanos Yirka Marina y Alexander (Meikel) por formar parte mí. Gracias por llenar mi vida de felicidad.*

*A mi nueva hermana Yordanys, por su amistad sincera y su presencia en mi vida. Gracias por enseñarme a enfrentar la vida desde otra perspectiva, por tus sabios consejos, por tu compañía, por ayudarme a seguir adelante. A mi primo Yader, a mis tíos Isdenys e Ismaida, por estar pendientes de mí en todo momento, por toda la ayuda que siempre me han ofrecido. A mi segunda madre Angela. Gracias por acogerme como un hijo, por ser incondicional y ayudarme a enfrentar los retos que la vida me impone. A mi segundo padre Alberto Villafaña. Gracias por enseñarme que aunque el trayecto parezca oscuro siempre hay una luz al final del camino. A mi compañero de tesis, por toda la ayuda brindada durante estos meses de intenso esfuerzo. Gracias por enseñarme que realmente eres una persona estupenda, este resultado de hoy es gracias a todo lo que nos hemos sacrificado. A mis tutores y en especial a Salvador. Gracias por formar responsabilidad en mí, sin usted el resultado de esta tesis sería imposible. A demás familiares, profesores y compañeros que de una forma u otra han estado presentes en estos largos años de estudio. Gracias a todos.*

*De Ediel Enrique*

*Primero que todo a mi madre y a mi padre, por haber estado siempre ahí para mí, por todo el apoyo, el cariño, la comprensión, la confianza que me han brindado y la dedicación de toda una vida.*

*A mi amada y luz en la vida, por brindarme todo su apoyo en los momentos más difíciles y a sus padres y hermano que son ya parte de mis seres más queridos.*

*A mi primo Evis que es el hermano que nunca tuve y me acompañó durante los tres primeros años de la carrera y de cierta manera nos servimos uno al otro, bueno, para eso está la familia. Te extrañé mi hermano.*

*A todos mis viejos amigos con los que tuve el placer de compartir estos 5 años y pasar junto a ellos momentos inolvidables y a las nuevas amistades adquiridas durante el transcurso de la carrera, todos ellos de alguna manera me ayudaron a ser mejor persona pues no hay nada más agradable que tener en quien confiar y desahogar en momentos difíciles.*

*A todo aquel que ha transitado por mi vida aportándome experiencia y fuerzas para cumplir mejor mis objetivos.*

*A mis tutores, por su apoyo, su comprensión, y paciencia.*

*De Alejandro Mariño.*

*A ti mamá porque esta tesis es tuya, como son tuyos todos mis logros, como es tuya mi vida.*

*A ti papá por no alejarte de mí nunca, por quererme tanto y ser un ejemplo para mí.*

*A mis hermanos, por ser los mejores del mundo.*

*De manera muy especial a dos estrellas que me iluminan desde el cielo:*

*A ti mi abuelita Paula, que la vida no permitió tenerte hoy aquí junto a mí, te AMO y te agradezco todo ese cariño que me diste siempre, te extraño mucho.*

*A ti abuelito Abilio, por cuidar de mí siempre, por estar al lado de mami en todo momento.*

*Dedico todos mis logros personales y profesionales a mis padres, a mis hermanos, a mi familia y a mis amigos. A ellos debo todo lo que soy y su amor me impulsa a seguir hacia adelante.*

*De Ediel Enrique*

*Dedico mis logros personales y profesionales a mi madre y mi padre, pues son los que más han alimentado mi vida con buenos pensamientos y deseos de ser alguien en la vida y a mi novia por ser la mujer más maravillosa del mundo y dedicar tanto amor y cariño hacia mi persona, por todo el apoyo brindado y por la sencilla razón de existir. Los amo.*

*De Alejandro Mariño*

La Universidad de las Ciencias Informáticas cuenta con centros de desarrollo de software dedicados a la producción de productos informáticos a nivel nacional e internacional, entre los que se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC). Para la elaboración de sus productos el departamento de Almacenes de Datos perteneciente a dicho centro, utiliza herramientas como “Visual Paradigm for UML” potenciando el uso de herramientas libres. La presente investigación surge a partir de la necesidad de realizar el diseño de las soluciones que se desarrollan en dicho departamento, ya que esta herramienta no provee la funcionalidad de modelar soluciones de almacenes de datos de forma dimensional. El presente trabajo tiene como objetivo desarrollar la extensión para la herramienta “Visual Paradigm for UML” que contribuya al perfeccionamiento del proceso de análisis y diseño de soluciones de almacenes de datos. Con esta extensión se pretende proveer a los especialistas del departamento una solución para realizar el modelado dimensional de las soluciones y de esta forma reducir el tiempo de desarrollo en la construcción de aplicaciones y garantizar un alto nivel en la calidad de las mismas.

Palabras claves: *modelo dimensional, plugin, Visual Paradigm for UML.*

## Abstract

The Universidad de las Ciencias Informáticas has software development centers dedicated to producing informatics products to national and international levels, among them the Centro de Tecnologías de Gestión de Datos (DATEC). For the development of its products the data warehouse's department, belonging to that center, uses tools such as " Visual Paradigm for UML " promoting the use of free tools. This research arises from the need for designing solutions develops in that department, as this tool does not provide the functionality to model solutions of dimensional data warehouses form. This paper aims to develop the extension to the " Visual Paradigm for UML " tool that contributes to improving the process of analysis and design of datawarehouse solutions. This extension aims at providing a solution to the specialists of the department for performing the dimensional modeling of the solutions and thus reduce the development time in building applications and ensure a high level of quality of the same.

Key words: Dimensional modeling, plugin, Visual Paradigm for UML.

# Índice de Contenido

# Índice de tablas

# Índice de figuras

# Introducción

Actualmente, en las actividades diarias de la mayoría de las organizaciones, se generan datos como producto secundario, que son el resultado de todas las transacciones que se realizan. Es muy común, que estos se almacenen y administren a través de sistemas en bases de datos. Gracias al auge de la ciencia y la tecnología es que los mismos han dejado de ser simples datos, y se han convertido en información que enriquece las decisiones de los usuarios. Precisamente, la Inteligencia de Negocios (Business Intelligence, BI por sus siglas en inglés), permite que el proceso de la toma de decisiones esté fundamentado sobre un amplio conocimiento de sí mismo y del entorno. Lo cual minimiza de esta forma el riesgo y la incertidumbre. Además propicia que las organizaciones puedan traducir sus datos en indicadores de estudio, y que estos se puedan analizar desde diferentes perspectivas. Con el fin de encontrar información que les permita construir modelos, mediante los cuales se podrían predecir eventos futuros.

La Inteligencia de Negocios, es un concepto que trata de englobar todos los sistemas de información de una organización para obtener de ellos no solo información o conocimiento, sino una verdadera inteligencia que le confiera a la organización una ventaja competitiva sobre sus competidores (Thomsen, 2002). La BI se apoya en un conjunto de herramientas que facilitan la extracción, la depuración, el análisis y el almacenamiento de los datos generados en una organización, con la velocidad adecuada para generar conocimiento y apoyar la toma de decisiones de los directivos y los usuarios oportunos (Vazquez, 2010), haciendo uso de herramientas que se basan principalmente en la utilización de Almacenes de Datos (AD).

Para referirse al Almacén de Datos como componente del proceso de Inteligencia de Negocios, Inmon señala que un Almacén de Datos es *"…una colección de datos orientada a un determinado ámbito (empresa, organización), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza"* (Inmon, 2002). Mientras que para Kimball *"…el Almacén de Datos es una colección de datos en forma de una base de datos que guarda y ordena información que se extrae directamente de los sistemas operacionales (ventas, producción, finanzas, marketing) y de datos externos*" (Kimball, 1998). Actualmente los AD poseen una filosofía para trabajar con modelos dimensionales o multidimensionales.

La Universidad de las Ciencias Informáticas[1] (UCI), surgió como un nuevo proyecto de la Revolución Cubana en el año 2002 (Salomón, 2014), con la misión de formar profesionales altamente calificados en la rama de la informática a partir de la vinculación estudio-trabajo como modelo de formación y contribuir al desarrollo de la Industria Cubana del Software. El modelo de producción en la UCI está estructurado en centros de desarrollo, vinculados directamente a las facultades con temáticas afines y especializados en diferentes áreas de investigación y desarrollo.

Uno de estos centros de desarrollo es el Centro de Tecnologías de Gestión de Datos (DATEC). El cual tiene como misión fundamental desarrollar productos y brindar servicios relacionados con las bases de datos y análisis de datos. Este centro precisamente se encarga de desarrollar soluciones que sirvan como apoyo a la toma de decisiones estratégicas como son: Almacenes de Datos, Cuadros de Mandos Integral, Tablero Digital (DashBoard) Corporativos, entre otros. El departamento de Almacenes de Datos, potenciando el uso de herramientas libres, utiliza la herramienta de modelado “Visual Paradigm for UML”, para guiar el análisis y diseño de las soluciones. Esta herramienta a pesar de las potencialidades que ofrece no cuenta con funcionalidades para el modelado de estructuras dimensionales, creación de cubos OLAP y vistas de análisis.

En el curso 2011-2012 se desarrolló una extensión de Visual Paradigm for UML para modelado dimensional en su versión 1.0 como resultado de un trabajo de diploma (Suárez, 2012). La cual permite crear el diseño lógico del AD a partir de un modelo dimensional, modelar estructuras dimensionales de un modelo lógico, diseñar el modelo lógico de la solución, y a partir de ahí generar el modelo físico. A pesar de la existencia de la extensión desarrollada, es imposible trabajar con las estructuras dimensionales de forma correcta. Cuando se desea crear el modelo lógico donde se establecen las relaciones entre las dimensiones y los hechos (modelo dimensional) con las tablas y campos de base de datos, a nivel relacional y se guarda el mismo, se pierden las relaciones existentes entre las clases. Igualmente al cargar el modelo de clases es necesario establecer nuevamente todas sus relaciones.

Como alternativa a este problema se decidió generar las relaciones en un fichero independiente. Esto trae consigo que a la hora de trabajar con el modelo se haga necesario contar con este fichero además del modelo de diseño de clases. Entorpeciendo el trabajo con la herramienta y tornándose engorrosa dicha

[1]Universidad de las Ciencias Informáticas, Km 2 ½ carretera San Antonio de los Baños, Torrens, Boyeros, Ciudad de la Habana, Cuba.

tarea. Además, la extensión aún no permite el diseño de los cubos ocasionando pérdidas de tiempo, mayor utilización de recursos humanos y requiere alto conocimiento por parte del usuario.

A partir de la situación descrita con anterioridad, se puede identificar el siguiente **problema de la investigación:** ¿Cómo contribuir al perfeccionamiento del proceso de análisis y diseño de soluciones de almacenes de datos en la herramienta "Visual Paradigm for UML?

El problema descrito centra el **objeto de estudio** en los modelos dimensionales para el proceso de análisis y diseño de soluciones de almacenes de datos, enmarcado en el **campo de acción** representación de modelos dimensionales utilizando la herramienta de modelado Visual Paradigm for UML.

Con la finalidad de darle cumplimiento al problema de la investigación se precisa el siguiente **objetivo general:** Desarrollar la extensión para la herramienta "Visual Paradigm for UML" que contribuya al perfeccionamiento del proceso de análisis y diseño de soluciones de almacenes de datos. En correspondencia con ello, se plantean como **objetivos específicos**:

1. Realizar un estudio del estado del arte de los conceptos, metodologías y herramientas para el desarrollo de la investigación.
2. Realizar el análisis y diseño de la extensión para modelar soluciones de almacenes de datos.
3. Realizar la implementación y pruebas de la extensión para modelar soluciones de almacenes de datos.

En la investigación se proponen un conjunto de **tareas de la investigación** que se describen a continuación para darle cumplimiento a los objetivos general y específicos:

1. Selección y revisión bibliográfica de la documentación técnica de "Visual Paradigm for UML" referida a la extensión de la herramienta.
2. Selección de la metodología, herramientas y tecnologías a utilizar en el desarrollo de la extensión para modelar soluciones de almacenes de datos.
3. Definición de los elementos para desarrollar modelos de datos dimensionales que serán modelados en "Visual Paradigm for UML" haciendo uso de perfiles.
4. Diseño de los artefactos generados por la metodología seleccionada para el diseño de la extensión.

5. Selección de patrones de diseño a emplear para el desarrollo de la extensión.
6. Implementación de la extensión para modelar soluciones de almacenes de datos.
7. Realización de pruebas a la extensión para modelar soluciones de almacenes de datos.
8. Documentación y corrección de las no conformidades identificadas en la ejecución de las pruebas.
9. Confección de los artefactos que propone el expediente de proyecto de la metodología seleccionada.

Para guiar las respuestas que se buscan con la presente investigación se identificaron las siguientes **preguntas científicas**:

1. ¿Qué tipo de problemas presenta la extensión en su primera versión que afectan el proceso de modelado de soluciones de almacenes de datos?
2. ¿Qué funcionalidades debe poseer la extensión para contribuir a al perfeccionamiento de las soluciones de almacenes de datos?
3. ¿Cómo se puede comprobar el funcionamiento de la extensión para modelar soluciones de almacenes de datos?

Durante el desarrollo de la investigación se utilizaron métodos teóricos y empíricos. Dentro de los métodos teóricos se emplearon los de análisis, el histórico lógico y de los métodos empíricos se utilizaron la observación, la entrevista y el análisis documental. A continuación se especifica el motivo de la selección de los mismos.

Los métodos teóricos empleados fueron:

**Método Analítico – Sintético:** a partir de la investigación realizada acerca del desarrollo de extensiones para VP en el departamento de Almacenes de Datos y las herramientas de desarrollo de software existentes, definir las tecnologías y metodología que serán utilizadas y arribar a las conclusiones de la investigación.

**Método Histórico-Lógico:** este método permite desarrollar el estudio del arte, previo al desarrollo de la investigación.

Para describir las características y obtener la información necesaria del objeto de estudio se utilizaron métodos empíricos, como los siguientes:

**Método de la observación:** se aplica para obtener el registro del avance del proyecto productivo y consignación de los acontecimientos pertinentes para la investigación. (Plan de observación ver **Anexo 1 en el expediente de proyecto**). Este método se utilizó en compañía de la técnica de entrevista.

**Entrevista no estructurada:** se aplica para obtener información sobre el desarrollo en el departamento de Almacenes de Datos de las extensiones para Visual Paradigm, las cuales son realizadas de forma individual a profesionales que están vinculados directamente con el desarrollo y evaluación de proyectos de este tipo así como para caracterizar el objeto de estudio.

**Análisis documental:** se utilizó para analizar, clasificar, verificar, seleccionar los contenidos en la bibliografía e informes de la investigación.

**Posibles resultados:** se espera obtener una extensión que se integre a la herramienta CASE Visual Paradigm for UML que permita modelar soluciones de almacenes de datos. Tal extensión beneficia el proceso de desarrollo de software en DATEC, particularmente al departamento de Almacenes de Datos. Se espera además que la documentación técnica generada por la investigación y la implementación de la extensión sirva de ayuda para posteriores implementaciones de extensiones de aplicaciones en el centro y la universidad.

La investigación se encuentra estructurada en tres capítulos que estarán guiados por los siguientes temas a tratar:

**Capítulo 1:** Fundamentos teóricos para el desarrollo de la “Extensión para modelar soluciones de almacenes de datos”. Referido al marco teórico y referencial de la investigación donde se realiza un estudio del estado del arte. Además, se describe el análisis y diseño de la misma y los principales aspectos de las herramientas a utilizar para la implementación de la aplicación.

**Capítulo 2:** Análisis y diseño de la extensión para modelar soluciones de almacenes de datos.

En este capítulo se describe la propuesta de solución para la situación problemática anteriormente planteada. Se definen las principales características que posee 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.

**Capítulo 3:** Implementación y pruebas de la extensión para modelar soluciones de almacenes de datos.

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

Cada capítulo es iniciado por una breve introducción donde se dan a conocer los temas que se desarrollarán. Para finalizar se presentan las conclusiones, las recomendaciones, las referencias bibliográficas, la bibliografía, un glosario de términos y el conjunto de anexos para un mejor entendimiento de lo expuesto en la investigación.

# Capítulo I: Fundamentos teóricos para el desarrollo de la "Extensión para modelar soluciones de almacenes de datos".

## 1.1 Introducción

Para comprender en qué consiste la presente investigación, se hace necesario conocer aspectos significativos relacionados con los modelos de datos, los modelos dimensionales y los modelos multidimensionales. Este capítulo constituye la base teórica de la solución propuesta. Es en esta temática donde se abordan varios conceptos, tecnologías y herramientas, que son objeto de análisis y estudio con el fin de fundamentar su uso en la solución que se propone.

## 1.2 Conceptos asociados al dominio del problema

Con el propósito de que el lector pueda alcanzar una mejor comprensión de los temas que serán tratados en este capítulo, directamente relacionados con el objeto de estudio de la investigación, se describen a continuación un grupo de conceptos asociados al dominio del problema.

### Modelo de datos

Un modelo de datos es un conjunto de mecanismos formales para representar y manipular información de manera general y sistemática: descripción de datos, operaciones y reglas de integridad (Decsai, 2013).

Estos describen los datos, sus relaciones, su significado y las condiciones que deben cumplir para reflejar correctamente la realidad deseada. Aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el desarrollo y uso de sistemas de información. Un modelo de datos es por tanto una colección de conceptos bien definidos matemáticamente que ayudan a expresar las propiedades estáticas y dinámicas de una aplicación (Ortiz, 2010).

### Modelo Entidad-Relación

Los diagramas o modelos entidad-relación (Entity Relationship Diagram, ERD por sus siglas en inglés) son una herramienta para el modelado de datos de un sistema de información. Expresan entidades relevantes para un sistema de información, sus inter-relaciones y propiedades. El modelo está compuesto por: entidades, atributos, relaciones, cardinalidad y llaves.

## Modelo dimensional

Básicamente el Modelo Dimensional (MDL en lo adelante) es el nombre que se le da a una técnica utilizada especialmente en AD. Este modelo difiere bastante del modelo Entidad-Relación que normalmente conocemos; aunque poseen la misma información, la organiza de forma diferente para garantizar la velocidad y eficiencia en la recuperación. El MDL busca presentar la información de una manera estándar, sencilla y sobre todo intuitiva para los usuarios, además permite que al acceso a la información sea mucho más rápido (López, 2012).

Este modelo le brinda facilidades al usuario para que puedan intuir el contenido de la información. Permite a las aplicaciones de inteligencia de negocios extraerla de forma rápida y sencilla. El "modelo dimensional" es el modo óptimo de organizar los datos en los sistemas de BI, y puede hacerse mediante el Procesamiento Analítico Relacional en Línea (Relational On Line Analytic Processing, ROLAP por sus siglas en inglés), o utilizando el Procesamiento Analítico Híbrido en Línea (Hybrid On Line Analytic Processing, HOLAP por sus siglas en inglés) (Urquizu, 2012). Para el modelo dimensional se utilizan diversos esquemas como el esquema estrella, el esquema copo de nieve (snowflake) y el esquema constelación de hechos.

El modelo dimensional divide el mundo de los datos en dos grandes conjuntos: las medidas y las descripciones del entorno de estas medidas. Generalmente son numéricas, se almacenan en las tablas de hechos y las descripciones de los entornos que son textuales se almacenan en las tablas de dimensiones.

***Hecho***

Se llama hecho a una operación que se realiza en el negocio la cual está estrechamente relacionada con el tiempo y es objeto de análisis para la toma de decisiones. También puede verse como un valor numérico que representa una actividad específica casi siempre con cifras que se suman entre sí.

***Dimensión***

Se conoce como dimensión a la característica de un hecho que permite su análisis posterior en el proceso de toma de decisiones y brinda una perspectiva adicional a un hecho dado.

Son agrupaciones lógicas de atributos con un significado común y atómico. Son usadas para seleccionar y agregar datos a un cierto nivel deseado de detalle. Viendo los datos dentro de un cubo se tiene la ventaja de que se puede manejar cualquier número de dimensiones. Sin embargo, usualmente un cubo tiene entre cuatro y doce dimensiones. Generalmente, un cubo soporta una vista de dos o tres dimensiones simultáneamente (Bernabue, 2009).

***Medida***

La medida es un dato numérico que representa una actividad específica de un negocio, mientras que una dimensión representa una perspectiva de los datos. Una medida contiene una propiedad numérica y una fórmula.

Existen tres clases de medidas:

- **Medidas aditivas:** pueden ser combinadas a lo largo de cualquier dimensión.
- **Medidas semiaditivas:** pueden ser combinadas a lo largo de una o más dimensiones.
- **Medidas no aditivas:** no pueden ser combinadas a lo largo de ninguna dimensión. (Bernabue, 2009).

***Granularidad***

Es el nivel más bajo de información que será almacenado en la tabla de hechos e indica el grado de detalle asociado a un hecho particular. La granularidad depende directamente del número de dimensiones que se asocian a la tabla de hechos. El primer paso para diseñar una tabla de hechos es determinar la granularidad. El gran factor y el más decisivo de la granularidad es el tiempo, ya que mientras menor sea el intervalo de tiempo, mayor será el grado de detalle obtenido. Se deben considerar otros factores como la carga del procesador, espacio de almacenamiento y satisfacer a cabalidad los requerimientos del cliente (Bernabue, 2009).

La estructura básica de un AD está definida por dos elementos fundamentales: esquemas y tablas, los cuales se describen a continuación con el fin de obtener un mejor entendimiento respecto al tema tratado en el presente trabajo de diploma.

***Tablas***

Un AD está conformado por varias tablas, en el modelo multidimensional existen dos tipos básicos de tablas:

***Tablas de Hechos***

Las tablas de hechos o tablas Fact, contienen los hechos que serán utilizados por los analistas de negocio para apoyar el proceso de toma de decisiones. Los hechos son datos instantáneos en el tiempo, que son filtrados, agrupados y explorados a través de condiciones definidas en las tablas de dimensiones (Bernabue, 2009). Estas tablas se encuentran relacionadas con sus respectivas tablas de dimensiones,

permitiendo que los hechos puedan ser accedidos, filtrados y explorados por los valores de los campos de estas tablas de dimensiones, obteniendo de este modo una gran capacidad analítica.

Existen dos tipos de hechos, los básicos y los derivados:

- **Hechos básicos:** son los que se encuentran representados por un campo de una tabla de hechos.
- **Hechos derivados:** son los que se forman al combinar uno o más hechos con alguna operación matemática o lógica y que también residen en una tabla de hechos. Estos poseen la ventaja de almacenarse previamente calculados, por lo cual pueden ser accedidos a través de consultas SQL sencillas y devolver resultados rápidamente, pero requieren más espacio físico en el AD.

***Tablas de Dimensiones***

Las tablas de dimensiones o tablas Lock-up definen cómo están los datos organizados lógicamente y proveen el medio para analizar el contexto del negocio. Representan los aspectos de interés, mediante los cuales el usuario podrá filtrar y manipular la información almacenada en la tabla de hechos (Bernabue, 2009).

Cada tabla de dimensión podrá contener los siguientes campos:

- Llave principal o identificador único.
- Llave foránea.
- Datos de referencia primarios: datos que identifican la dimensión. Por ejemplo: nombre del cliente.
- Datos de referencia secundarios: datos que complementan la descripción de la dimensión. Por ejemplo: e-mail del cliente, fax del cliente.

## 1.3 Modos de almacenamiento de datos

Los modos de almacenamiento de datos caen en la categoría del procesamiento analítico en línea, que de este se derivan el procesamiento relacional, el multidimensional y el procesamiento híbrido, que no es más que la combinación del procesamiento relacional y el procesamiento multidimensional.

***Procesamiento Analítico en Línea (OLAP)***

El Procesamiento Analítico en Línea (On Line Analytic Processing, OLAP por sus siglas en inglés), es el componente más poderoso de los AD, ya que es el motor de consultas especializado del Almacén de Datos. En estos modelos, los datos son vistos como cubos los cuales consisten en categorías descriptivas (dimensiones) y valores cuantitativos (medidas).

Este procesamiento permite el uso más eficiente de los AD en línea, proporcionando respuestas rápidas a consultas analíticas complejas e iterativas utilizadas generalmente para sistemas de ayuda a la toma de decisiones. Presenta los datos de manera natural, de forma que los usuarios puedan entender mejor la información. Con la utilización del Procesamiento Analítico en Línea se puede ver un conjunto de datos del negocio de muchas y diversas formas sin mucho esfuerzo. Los archivos OLAP o cubos modelan los datos en dimensiones (Kimball, 2002).

Las principales características de OLAP son:

- **Rápido:** proporciona la información al usuario a una velocidad constante (mayormente en cinco segundos o menos).
- **Análisis:** realiza análisis estadísticos y numéricos de los datos de forma básica.
- **Compartida:** permite compartir los datos potencialmente confidenciales a través de una gran cantidad de usuarios, implementando para esto los requerimientos de seguridad necesarios.
- **Multidimensional:** permite ver la información en determinadas vistas o dimensiones.
- **Información:** accede a todos los datos necesarios, donde quiera que estos residan, y mientras no esté limitada por el volumen (Beltrá, 2007).

Los sistemas OLAP se clasifican en las siguientes categorías (Zorrilla, 2008)

***Procesamiento Analítico Relacional en Línea (ROLAP)***

El Procesamiento Analítico Relacional en Línea (Relational On Line Analytic Processing, ROLAP por sus siglas en inglés) cuenta con todos los beneficios de un SGBD Relacional a los cuales se les provee extensiones y herramientas para poder utilizarlo como un Sistema Gestor de BD (Bernabue, 2009).

Accede directamente a los datos del AD y soporta técnicas de optimización de accesos para acelerar las consultas, por ejemplo: partición de datos a nivel de aplicación, soporte a la desnormalización y joins múltiples. Esta organización tiene como desventaja que es más lenta que las demás estrategias de almacenamiento (Sinnexus, 2012).

Las principales características de ROLAP son: almacena la información de forma relacional, posee tres capas lógicas: de almacenamiento, de análisis y de presentación. Utiliza índices de mapas de bits, utiliza índices de Join, posee técnicas de particionamiento de datos, posee optimizadores de consultas y cuenta con extensiones del SQL (drill-up, drill-down).

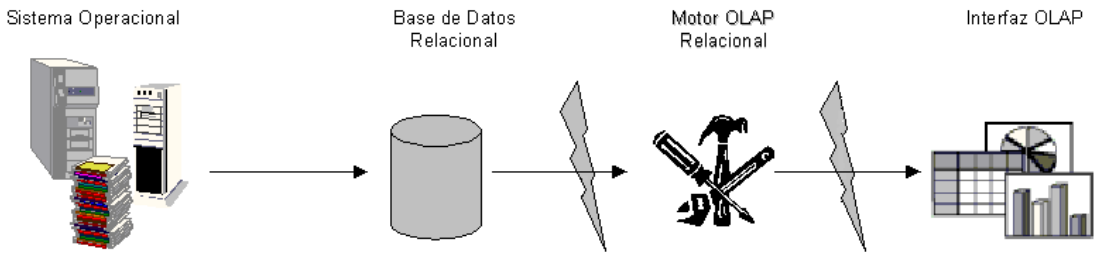


**Fig 1. Modelo de almacenamiento ROLAP. Fuente (Fernández, 2013)**

***Procesamiento Analítico Multidimensional en Línea (MOLAP)***

El Procesamiento Analítico Multidimensional en Línea (Multidimentional On Line Analytic Processing MOLAP, por sus siglas en inglés) tiene como objetivo almacenar físicamente los datos en estructuras multidimensionales de manera que la representación externa y la interna coincidan. Para ello, se dispone de estructuras de almacenamiento específicas (Arrays) y técnicas de compactación de datos que favorecen el rendimiento del depósito de datos.

Usa las bases de datos multidimensionales bajo la premisa que OLAP está mejor implantado almacenando los datos multidimensionalmente. Tiene el mejor tiempo de respuesta, excelente rendimiento y compresión de datos. Tiene la desventaja que es una solución particular, para soluciones con volúmenes de información y cantidad de dimensiones más bien modestos. (Sinnexus, 2012)

**Las principales características de MOLAP son: p**osee tecnología optimizada para consultas y análisis, basada en el modelo multidimensional. Cuenta con un motor especializado. Provee herramientas limitadas y propietarias. No es adecuada para muchas dimensiones, construye y almacena datos en estructuras multidimensionales.

Almacenamiento en estructura multidimensional.Mayor rapidez de respuestas.

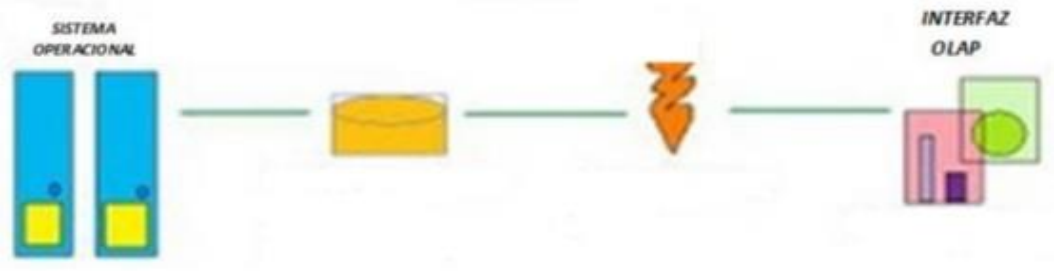


**Fig 2. Modelo de almacenamiento MOLAP. Fuente (Fernández, 2013).**

***Procesamiento Analítico Híbrido en Línea (HOLAP).***

El Procesamiento Analítico Híbrido en Línea (Hybrid On Line Analytic Processing, HOLAP por sus siglas en inglés) constituye un sistema híbrido entre MOLAP y ROLAP, que combina estas dos implementaciones para almacenar algunos datos en un motor relacional y otros en una base de datos multidimensional, para brindar una solución con las mejores características de ambas: desempeño superior y gran escalabilidad.

Mantiene los registros de detalle en la base de datos relacional y las agregaciones en un almacén MOLAP separado. Es usado generalmente para cubos que requieren rápida respuesta con una gran cantidad de datos (Sinnexus, 2012).

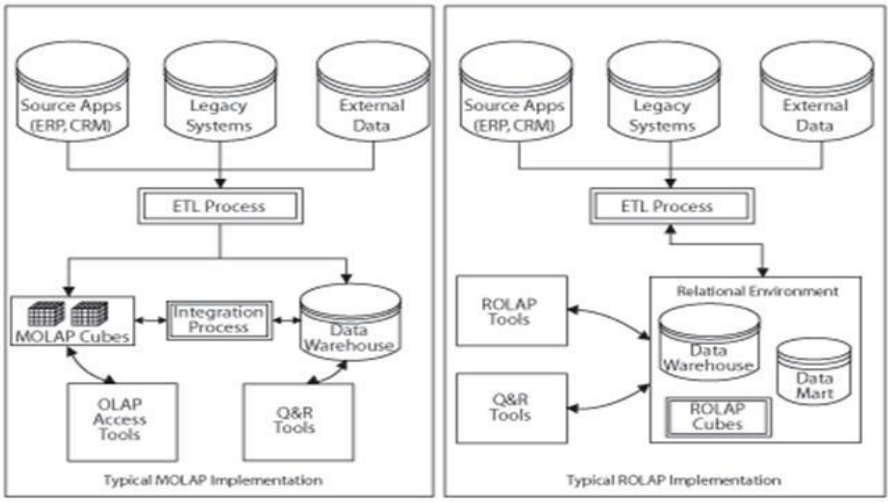


**Fig 3. Modelo de almacenamiento HOLAP. Fuente (Fernández, 2013).**

A continuación se presenta una tabla donde se muestran las ventajas y las desventajas de las categorías del Procesamiento Analítico en Línea (OLAP)

**Tabla 1. Ventajas y Desventajas de las categorías del OLAP. (Elaboración propia).**

| Categoría | Ventajas | Desventajas | Categoría |
|---|---|---|---|
| ROLAP | Ahorra espacio de almacenamiento. Útil cuando se trabaja con grandes conjuntos de datos. | El tiempo de respuesta a consultas es mayor. | ROLAP |
| MOLAP | Mejor rendimiento en los tiempos de respuesta. | Duplica el almacenamiento de datos (ocupa más espacio). | MOLAP |
| HOLAP | Buen tiempo de respuesta sólo para información resumida. | Volúmenes de datos más grandes en la BD relacional. | HOLAP |

## 1.4 Esquemas

La colección de tablas en el almacén se conoce como esquema. Los esquemas caen dentro de dos categorías básicas: esquema de estrellas y esquema copo de nieve. De estos, se deriva un tercer esquema nombrado esquema constelación de hechos.

***Esquema estrella***

El esquema de estrellas (Star Schema) deriva su nombre del hecho que su diagrama forma una estrella, con puntos radiales desde el centro (Herrera, 2007).

Este esquema consta de una tabla de hechos central y de varias tablas de dimensiones relacionadas a esta, a través de sus respectivas claves. En la siguiente figura se puede apreciar un esquema de estrellas estándar:

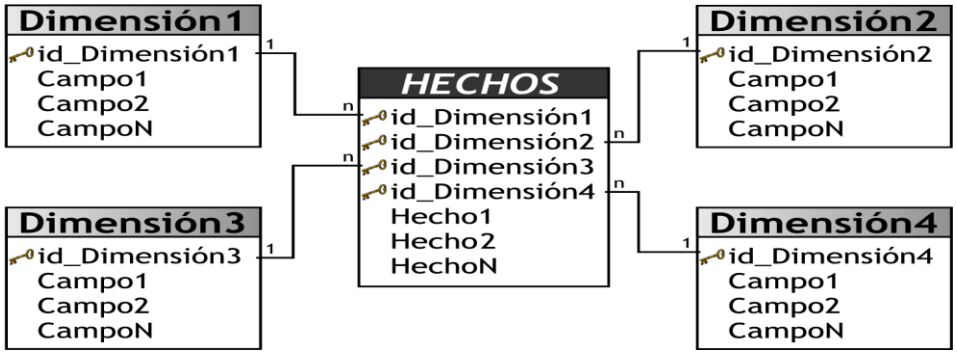


**Fig 4. Esquema de estrella con una sola tabla de hechos con enlaces a varias tablas de dimensiones. Fuente (Bernabue, 2009).**

El esquema de estrellas es el más simple de interpretar y optimiza los tiempos de respuesta ante las consultas de los usuarios. Este modelo es soportado por casi todas las herramientas de consulta y análisis, y los metadatos son fáciles de documentar y mantener, sin embargo es el menos robusto para la carga y es el más lento de construir. Un esquema de estrellas puede tener cualquier número de tablas de dimensiones (IBM, 2013).

**Características del esquema de estrellas:** posee los mejores tiempos de respuesta, su diseño es fácilmente modificable. Existe paralelismo entre su diseño y la forma en que los usuarios visualizan y manipulan los datos. Simplifica el análisis y facilita la interacción con herramientas de consulta y análisis (Bernabue, 2009).

***Esquemas de copo de nieve***

El esquema de copo de nieve (Snowflake Schema) representa una extensión del modelo en estrella cuando las tablas de dimensiones se organizan en jerarquías de dimensiones. Este modelo es más cercano a un modelo de entidad relación, que al modelo en estrella, debido a que sus tablas de dimensiones están normalizadas.

Este tipo de modelo, permite segregar los datos de las tablas de dimensiones y proveer un esquema que sustente los requerimientos de diseño. Es muy flexible y puede implementarse después de que se haya desarrollado un esquema en estrella.

Este esquema consta de una tabla de hechos que está conectada a muchas tablas de dimensiones, que pueden estar conectadas a otras tablas de dimensiones a través de una relación de muchos a uno (IBM, 2013). En la siguiente figura se puede apreciar un esquema de copo de nieve:

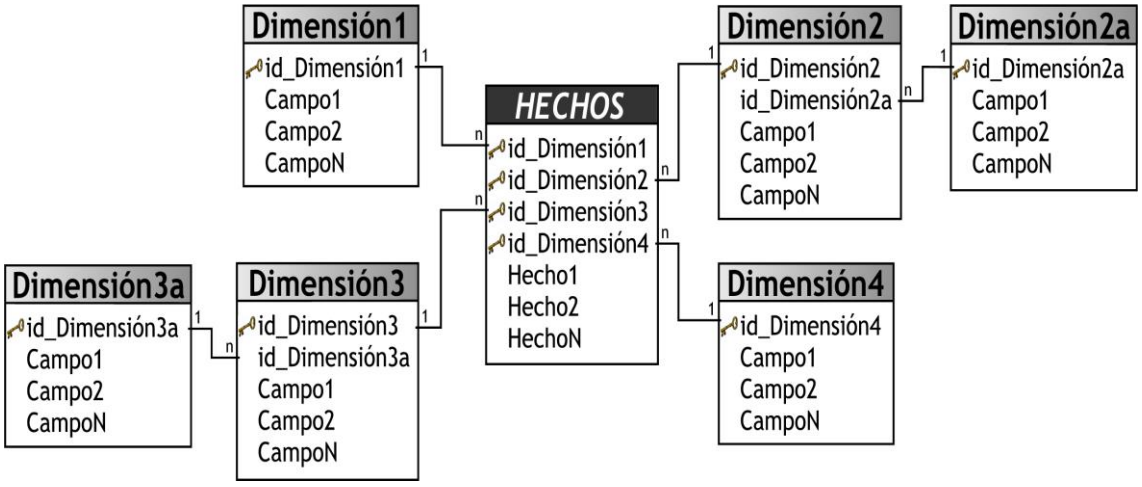


**Fig 5. Esquema de copo de nieve. Fuente** (Bernabue, 2009)**.**

Como se puede apreciar en la figura anterior, existe una tabla de hechos central que está relacionada con una o más tablas de dimensiones, quienes a su vez pueden estar relacionadas o no con una o más tablas de dimensiones.

**Características del esquema de copo de nieve:** posee mayor complejidad en su estructura, hace una mejor utilización del espacio. Es muy útil en tablas de dimensiones de muchas tuplas. Las tablas de dimensiones están normalizadas, por lo que requiere menos esfuerzo de diseño y puede desarrollar

clases de jerarquías fuera de las tablas de dimensiones, que permiten realizar análisis de lo general a lo detallado y viceversa.

A pesar de todas las características y ventajas que trae aparejada la implementación del esquema copo de nieve, existen dos grandes inconvenientes de ello: si se poseen múltiples tablas de dimensiones, cada una de ellas con varias jerarquías, se creará un número de tablas bastante considerable, que pueden llegar al punto de ser inmanejables y al existir muchas uniones y relaciones entre tablas, el desempeño puede verse reducido (Bernabue, 2009).

***Esquema Constelación de hechos***

Un esquema de constelación de hechos (Starflake Schema) es una combinación de un esquema de estrella y un esquema de copo de nieve. Los esquemas de constelación son esquemas de copo de nieve en los que sólo algunas de las tablas de dimensiones se han desnormalizado. El objetivo de este esquema es aprovechar las ventajas de los esquemas de estrella y de copo de nieve (IBM, 2013). Las jerarquías de los esquemas de estrella están desnormalizadas, mientras que las jerarquías de los esquemas de copo de nieve están normalizadas (Herrera, 2007). En la siguiente figura se puede apreciar un esquema constelación de hechos:

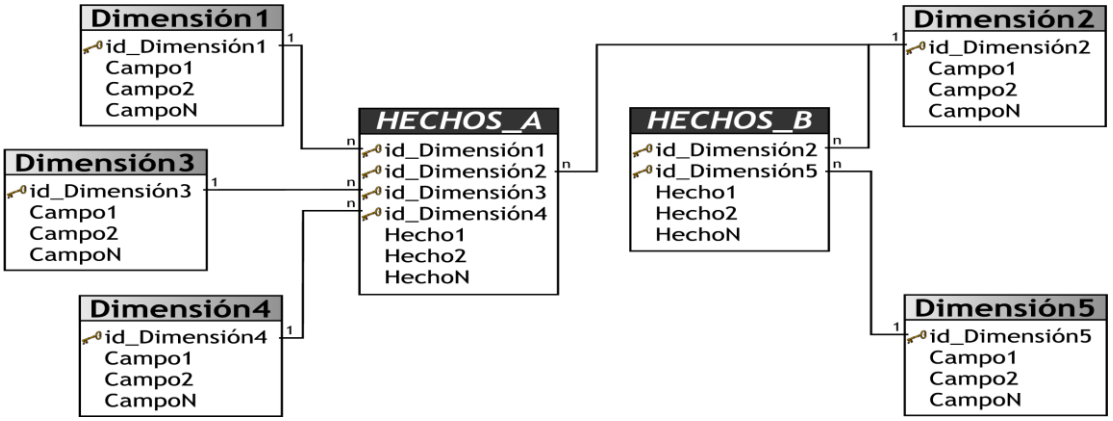


**Fig 6. Esquema constelación de hechos. Fuente (Bernabue, 2009).**

**Características del esquema constelación de hechos:** permite tener más de una tabla de hechos, por lo cual se podrán analizar más aspectos claves del negocio con un mínimo esfuerzo adicional de diseño. Contribuye a la reutilización de las tablas de dimensiones, ya que una misma tabla de dimensión puede utilizarse para varias tablas de hechos. No es soportado por todas las herramientas de consulta y análisis (Bernabue, 2009).

Los tres esquemas anteriores pueden ser implementados basándose en el modelado dimensional, para ello pueden utilizarse los cubos multidimensionales, que son una de las estructuras más completas para representar los datos.

Para un mejor entendimiento de los esquemas explicados anteriormente se aborda el concepto de cubo multidimensional por estar fuertemente asociado al modelo dimensional:

## 1.5 Cubos multidimensionales

Un cubo multidimensional es un conjunto formado por todas las tablas de dimensiones y la tabla hecho. Al final dan una vista en forma de cubo cuyas celdas están compuestas por las medidas de la tabla hecho. Permiten que los reportes sean obtenidos con un bajo tiempo de respuesta y que el análisis de los datos sea diverso. Pues cada cara del cubo se refiere a un análisis distinto de las medidas almacenadas (Gallardo, 2012). Unas de las estructuras más utilizadas para representar los datos del AD es el cubo multidimensional. Este convierte los datos que se encuentran en las filas y columnas, en una matriz de N dimensiones.

*"Los cubos son elementos claves en OLAP (On Line Analytical Processing), una tecnología que provee rápido acceso a datos en un Almacén de Datos (Data Warehouse). Los cubos proveen un mecanismo para buscar datos con rapidez y tiempo de respuesta uniforme independientemente de la cantidad de datos en el cubo o la complejidad del procedimiento de búsqueda."* (Gallardo, 2012).

Entre los objetos más importantes a incluir en el cubo están los siguientes:

- Indicadores: son sumarizaciones efectuadas sobre hechos o expresiones que posibilitan analizar los datos almacenados, pertenecen a las tablas de hechos.
- Atributos: constituyen los criterios de análisis que se utilizarán para analizar los indicadores, pertenecen a las tablas de dimensiones. Dentro de un cubo multidimensional, los atributos son los ejes del mismo.
- Jerarquías: representa una relación lógica entre dos o más atributos.

**Las jerarquías poseen las siguientes características:** pueden existir varias en un mismo cubo. Están compuestas por dos o más niveles. Se tiene una relación "1-n" o "padre-hijo" entre atributos consecutivos de un nivel superior y uno inferior.

En un cubo multidimensional, los atributos existen a lo largo de varios ejes o dimensiones. La intersección de las mismas representa el valor que tomará el indicador que se está evaluando. En la siguiente figura se puede apreciar un cubo multidimensional:

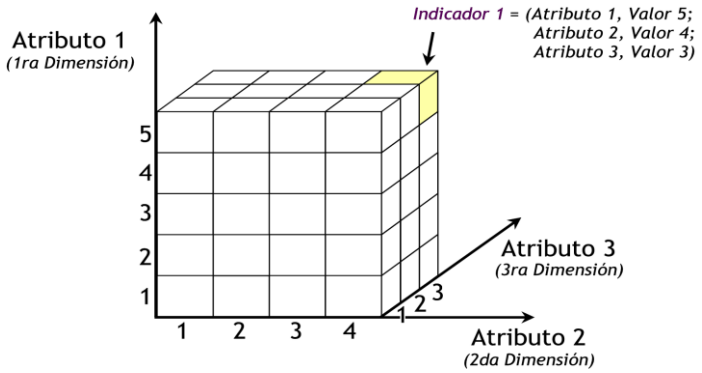


**Fig 7. Cubo Multidimensional**

## 1.6 Soluciones existentes para modelar almacenes de datos

En la actualidad existen varias soluciones que permiten a los desarrolladores modelar almacenes de datos, algunas recientes y con no mucha aceptación, al tiempo que otras sobresalen por su facilidad de uso.

***Algunas de las soluciones existentes en el mundo***

En el mundo existen diversas empresas que se dieron a la tarea de implementar soluciones para modelar almacenes de datos. Las más destacadas actualmente son las ofrecidas por ORACLE y por ERWIN. Estas soluciones ofrecen robustez y calidad en los productos obtenidos; pero hoy presentan un gran inconveniente que frena su empleo. Se distribuyen bajo licencia privativa. El cliente tiene que pagar para hacer uso de ellas, lo cual constituye una barrera que frena su empleo. Como solución a ello en Cuba se han ido desarrollando algunas soluciones que resuelven grandes dificultades.

***Algunas de las soluciones existentes en Cuba***

En Cuba existen diferentes entidades que han desarrollado extensiones de la herramienta de modelado Visual Paradigm for UML para facilitar el trabajo con la misma y con el objetivo de agregar nuevas funcionalidades que han resultado necesarias poder realizar en dicha herramienta. Ejemplo de ello se evidencia en la Empresa de Tecnologías de la Información para la Defensa, XETID que está llevando a cabo el desarrollo de una plataforma de BI con el objetivo de ayudar a la toma de decisiones en la

empresa, lograr la soberanía tecnológica y además de integrar el Sistema de Gestión Empresarial en Cuba (ERP).

Dentro de la plataforma se desarrolló una aplicación (Diseñador de Esquemas), como resultado de un trabajo de diploma (Medina, 2012) que sustituye el trabajo con el esquema Workbench, facilitando el diseño de los cubos OLAP. Esta plataforma brinda muchas facilidades pero está desarrollada para su entorno de trabajo. Está basada en la gestión de proyectos, en la planeación estratégica y en un marco de trabajo específico para las herramientas y soluciones que este centro genera. Por lo que su adaptación se hace engorrosa a otros entornos de trabajo.

En la actualidad, la extensión de Visual Paradigm desarrollada por el centro de desarrollo DATEC, a pesar de permitir diseñar el modelo lógico del almacén, generar el modelo físico y obtener la estructura básica de los cubos OLAP; presenta un marcado número de deficiencias para su despliegue. Imposibilitando su uso por parte de los especialistas del departamento. Por lo que surge el presente trabajo de diploma.

Dicha extensión presenta como objetivo general: Desarrollar una extensión de la herramienta “Visual Paradigm for UML” para modelar estructuras de datos dimensionales. La misma está constituida por una serie de funcionalidades con el objetivo de brindar a los usuarios pasos automatizados en el proceso de creación de los almacenes. Dichas funcionalidades son:

1. Modelar lógicamente la estructura de un AD.
2. Representar las jerarquías en el modelado del AD.
3. Generar el modelo físico a partir del modelo lógico.
4. Generar un XML con el diseño preliminar de los cubos OLAP a partir del diseño del AD.
5. Generar el script para cargar el diseño en el gestor de BD.

Sin embargo esta herramienta hoy presenta dificultades en su funcionamiento como son las que se listan a continuación:
No permite diagramar correctamente el diseño lógico del AD

- Permite que un hecho se relacione con otro hecho.
- No permite la relación de un hecho con una dimensión de uno a uno y de uno a muchos, en cambio genera relacionado con el mismo una entidad lógica.
- La entidad dimensional de tipo puente está conceptualmente mal utilizada pues se relaciona entre un hecho y una dimensión y recibe la llave primaria del hecho.

- Al cerrar y abrir la aplicación se pierden las relaciones existentes entre las entidades dimensionales y visualmente queda desordenado.
- Las vistas no muestran la información verdadera que contiene el diagrama.
- No genera los cubos en el esquema.
- En ocasiones funcionalidades que se realizan correctamente bajo la lógica implementada dejan de funcionar y se comportan de manera extraña.
- Cuando se genera el diagrama entidad relación, las jerarquías no son generadas y en el caso del puente no queda bien representado siguiendo la lógica implementada.

## 1.7 Desarrollo de extensiones para Visual Paradigm for UML

La implementación de extensiones para aplicaciones constituye un mecanismo para la incorporación de funcionalidades que la aplicación no provee a los usuarios. 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 asocia la palabra extensión con plugin o componente.

Una extensión o generalmente llamada plugin es un módulo de software o de hardware que añade características o funcionalidades específicas a una herramienta más grande. En nuestros días es muy común ver cómo las grandes aplicaciones sufren cambios por parte de los programadores, ya que las técnicas de trabajo van evolucionando y se necesita de nuevas alternativas que las mismas no pueden brindar.

Visual Paradigm for UML 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. A su vez cuenta con los medios para extender funcionalidades que no posee, dando soporte a las extensiones de la aplicación. Los programadores pueden implementar, reutilizar sus clases e interfaces ya que VP provee de forma libre una interfaz de programación API[2] que le permite esta integración. Así se desarrollan funciones agregadas para las necesidades específicas de cada usuario.
La 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. Propone que es importante definir el

[2] API (del inglés Application Programming Interface) o Interfaz de programación de aplicaciones (IPA). Es una interfaz de comunicación entre componentes de software. Consiste en proporcionar un conjunto de funciones de uso general, que son llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación.

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 y formas o conectores personalizados del plugin.

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 en una carpeta específica llamada plugin en el directorio de instalación de VP, en caso de no existir el directorio es necesario crearlo. La realización del proceso de despliegue se explica de forma detallada en el **CAPITULO 2, epígrafe 2.3.1.**

## 1.8 Proceso de desarrollo de Software

El proceso de desarrollo de software no es una tarea sencilla. Muchos proyectos de los que se construyen presentan una serie de problemas ya que tienen importantes retrasos, o simplemente no cumplen con las expectativas del cliente. Para no incurrir en estas dificultades, la presente investigación realiza un estudio sobre los distintos tipos de metodologías de desarrollo, seleccionando de ellas la que mejor se adapta a las exigencias y condiciones de la solución propuesta. Unido a esto se abordan el lenguaje de programación y herramienta CASE seleccionada para modelar los diferentes diagramas que sirven como soporte y documentación ingenieril en el proceso de desarrollo.

***Metodologías de desarrollo del software***

Actualmente a la hora de construir un software la incorrecta planificación es frecuentemente el problema que más se presenta. Como solución a esta problemática surgen las metodologías de desarrollo de software.

En el libro El Proceso Unificado de Desarrollo de Software, autores como Jacobson, Rumbaugh y Booch plantean que una metodología es un proceso, que aplicado al mundo del desarrollo del software se utiliza para definir quién debe hacer qué, cuándo y cómo debe hacerlo. (Jacobson, 2010) Existen fundamentalmente dos tipos de metodologías para contribuir al desarrollo de software: las metodologías tradicionales, también conocidas como metodologías pesadas y las metodologías ágiles o ligeras.

Entre las metodologías de desarrollo más conocidas están Programación Extrema (Extreme Programming, XP), AUP (Agil Unified Process), SCRUM 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.

Las metodologías ágiles dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas, en estas el cliente llega a formar parte del equipo de trabajo. Las metodologías tradicionales se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, además de las herramientas y notaciones que se usarán.

***Programación Extrema (XP)***

*"Muchas de las prácticas propuestas contribuyen a maximizar la comunicación entre las personas, permitiendo de esa forma una mayor transferencia de conocimiento entre los desarrolladores y con el cliente, quien también es parte del equipo"* (Hernán, 2009).

La metodología de desarrollo Programación Extrema (Extreme Programming, XP por sus siglas en inglés), fue concebida por Kent Beck (Ver anexo 1), como un proceso de creación de software diferente al convencional. Según Beck "*XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar software*" (Beck, 2009). Esta se ha convertido en la actualidad en la metodología ágil para el desarrollo de software más exitosa, pues cuenta con la aceptación de la mayoría de los equipos de desarrollo. Se utiliza en proyectos con pequeños equipos de desarrollo y con corto plazo de entrega. Se basa en la retroalimentación entre el cliente y el equipo de desarrollo, buena comunicación entre los participantes y simplicidad en las soluciones implementadas. Consiste en una programación rápida, cuya particularidad es tener como miembro del equipo al usuario final. Es adecuada para proyectos con requisitos imprecisos, muy cambiantes, y donde existe un alto riesgo técnico. (Jeffries, 2000) Incluye buenas prácticas de desarrollo como son el Desarrollo Guiado por Test, (Test Development Driver, TDD por sus siglas en inglés) (Marches, 2002) e Integración Continua (Sanchez, 2004).

**Entre sus principales ventajas están**: 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 y el cliente o el usuario se convierten en miembro del equipo (ESC02).

**Entre las características de esta metodología destacan:** desarrollo iterativo e incremental, programación en parejas. Pruebas unitarias continuas, integración del equipo de desarrollo con el cliente y refactorización del código.

**La solución propuesta presenta las siguientes particularidades:**

- El hecho de que el cliente (los miembros del departamento), se encuentre físicamente disponible, permite involucrarlo en el desarrollo de la solución y contribuye a la retroalimentación directa.
- A pesar de su complejidad, se trata de un proyecto de pequeñas dimensiones, que requiere de ejemplos de uso y código funcional, pero no de documentación exhaustiva.
- La captura de requisitos mediante Historias de Usuario resulta efectiva en este caso, pues las descripciones de las mismas serán realizadas por programadores del departamento, lo cual propicia un mejor entendimiento entre los desarrolladores y el cliente.
- "El diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación." (Kent Beck) La estrecha relación entre el equipo de desarrollo y el cliente, permite que se aplique este tipo de comunicación, posibilitando que no se deban generar grandes volúmenes de documentación.
- El equipo de desarrollo está compuesto por dos personas, lo cual propicia la revisión y discusión del código mientas se escribe. Esta práctica tributa a la calidad final del código y es defendida por XP.

**En este escenario, el uso de esta metodología supone:**

- Rapidez en la obtención de versiones a causa de la rápida codificación.
- Obtención de productos fiables y robustos debido al diseño de las pruebas previo a la codificación.
- Reducción del número de posibles errores debido a que los requisitos del software son fáciles de modificar.
- Énfasis en la adaptabilidad y no en la previsibilidad, por lo que los cambios en los requisitos no afectan los planes de entrega del producto.
- Mínimo de documentación.

***Roles que propone la metodología seleccionada***

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 (Guerrero, 2011).

Teniendo en cuenta las necesidades y las características del presente trabajo de diploma los roles definidos, de acuerdo con la propuesta original de Kent Beck para el desarrollo de la solución serán los siguientes:

**Programador**

El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo. En 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.

**Cliente**

El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

**Encargado de pruebas (Tester)**

El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Es responsable de hacer funcionar las pruebas regularmente y comunicar sus resultados al resto del equipo. Debe ser una persona con capacidad de abstracción, mucho tacto y facilidad de comunicación, dado que está en un lugar intermedio entre los programadores y el cliente.

***Lenguaje de programación***

Autores como Terry definen el término de lenguaje como: *"... número finito de cadenas sobre un alfabeto que puede ser definido listando todas las cadenas o dando una regla para su derivación"*. Específicamente, los lenguajes de programación (LP, en lo adelante) se ajustan a la definición antes mencionada, aunque incluyen muchas más restricciones, donde las cadenas no son más que las palabras reservadas y los identificadores. Los LP pueden ser de propósito general o específico, los primeros, como su nombre lo indican pueden usarse para el desarrollo de "cualquier" tipo de software; los segundos, sólo para un ámbito específico.

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.

**Java**

Java es un lenguaje de programación de plataforma independiente desarrollado por Sun Microsystems en los años 90. Emplea el paradigma de programación orientado a objeto para propósito general. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel. Fue diseñado para crear software altamente fiable, para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Su principal característica es la de ser un lenguaje compilado e interpretado, la cual lo ha llevado a niveles muy alto en la preferencia de los desarrolladores de la comunidad internacional.

**Entre sus principales ventajas están:** es un lenguaje orientado a objetos, es multiplataforma. Posee capacidad multihilo, gran rendimiento y permite la creación de aplicaciones distribuidas.
Para la implementación de la extensión se empleó este lenguaje, por ser el lenguaje de programación propuesto por la API de desarrollo de la herramienta “Visual Paradigm for UML“, teniendo en cuenta las características que presenta el mismo.

***Herramientas a utilizar***

Para llevar a cabo el desarrollo de la extensión que responda a los requerimientos de esta investigación se hace necesario el uso de un conjunto de herramientas cuyas funcionalidades respondan a los lineamientos, principios y teorías de: metodología, paradigmas y lenguajes que han sido seleccionados, de modo tal que las mismas contribuyan a obtener un producto de calidad como lo señalara Pressman. Las premisas de mayor importancia para su selección son: políticas de migración de la UCI hacia el software libre y la capacidad que tienen para funcionar sobre varias plataformas.

**Herramienta CASE**

Las herramientas de Ingeniería de Software Asistido por Computadora (Computer Aided Software Engineering, CASE por sus siglas en inglés) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de un software reduciendo su costo. Pueden servir de ayuda durante el ciclo

de vida de desarrollo del software en tareas como el diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente a partir del diseño dado, compilación automática, documentación o detección de errores. Existen múltiples herramientas con estos fines, tales como Enterprise Architect, Visual Paradigm, ArgoUML, Rational Rose, entre otras.

**Visual Paradigm 8.0**

Visual Paradigm es una herramienta CASE profesional que da soporte al análisis y diseño orientado a objetos, construcción, pruebas y despliegue del ciclo de vida completo del desarrollo de software.Tiene licencia dual, gratuita y comercial. Soporta las últimas versiones de UML y la Notación y Modelado de Procesos de Negocios. Brinda la posibilidad de crear todos los tipos de diagramas de clases, código inverso, generación de código a partir de diagramas y generar documentación (paradigm, 2010).

**Entre sus principales ventajas están:**

- **Multiplataforma:** soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
- **Interoperabilidad:** intercambia diagramas UML y modelos con otras herramientas. Soporta la importación y exportación a formatos XMI[3], XML[4] y archivos Excel. Permite importar proyectos de Rational Rose y la integración con Microsoft Office Visio5.
- **Integración con Entornos de Desarrollo:** apoyo al ciclo de vida completo de desarrollo de software en IDE como: Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, Jbuilder y otros.

El sitio oficial del Visual Paradigm enumera otro conjunto de características como: permitir exportar la documentación a varios formatos incluidos MS Word y PDF, comparación entre diagramas resaltando las diferencias, diseño de patrones y trazabilidad en el desarrollo de los modelos, soporta el proceso de ingeniería de varios gestores de base de datos

Su versión 8.0 incluye la funcionalidad de crear y especificar perfiles UML, la cual resulta de vital importancia para la implementación y ejecución de extensiones para la herramienta. Debido a todas las características mencionadas y los beneficios que brinda para el desarrollo de software, especialmente

[3] XMI es un estándar de la OMG (Object Management Group) para el intercanvio de ficheros UML en formato XML.

[4] XML: lenguaje de marcado extensible (*e**X**tensible **M**arkup **L**anguage,* por sus siglas en inglés), que permite definir la gramática de lenguajes específicos, siendo útil cuando varias aplicaciones se deben comunicar entre sí.

referentes al modelado, se decidió utilizar Visual Paradigm for UML 8.0 para el modelado de la extensión, además de ser la herramienta de modelado escogida por la Universidad para el desarrollo de software, asumida por el centro DATEC en sus producciones.

**Schema-Workbench**

Schema Workbench es la herramienta gráfica de diseño que permite la construcción de los esquemas de cubos OLAP visualmente y además permite publicarlos al servidor inteligencia del negocio para que puedan ser utilizados en los análisis por los usuarios de la plataforma.

Este programa publicado en el año 2007, entrega todas las facilidades para poder realizar el modelo lógico del cubo OLAP al cual se le realizarán las consultas. Este se conecta directamente con la BD para así poder diseñar los cubos OLAP que se requieren para que el usuario final pueda visualizar los indicadores (Espinosa, 2010).

El schema workbench ofrece las siguientes funcionalidades: editor de esquemas integrados con un origen de datos subyacente para su validación. Prueba de consultas MDX[5] contra el esquema y la BD.
Por lo tanto Schema Workbench es una herramienta gráfica para el diseño de los cubos multidimensionales. El motor de Mondrian procesa peticiones de MDX que permiten configurar el cubo en dependencia de los intereses. Genera un archivo XML que registra todas las acciones realizadas con la herramienta.

**Pentaho BI Server**

Pentaho BI Server es una plataforma que provee el soporte y la infraestructura necesarios para crear soluciones de BI. Proporciona los servicios básicos, incluidos autenticación, registro, auditoría, servicios web y motor de reglas. También incluye un motor de solución que integra reportes, análisis, tableros de comandos y componentes de minería de datos.
Pentaho BI Server, funciona como un sistema basado en administración web de informes, el servidor de integración de aplicaciones y un motor de flujo de trabajo ligero (secuencias de acción.) Está diseñado para integrarse fácilmente en cualquier proceso de negocio (Summan, 2014).

[5] Las expresiones multidimensionales (MDX) permiten consultar objetos multidimensionales, como los cubos, y devolver conjuntos de celdas multidimensionales que contengan los datos del cubo.

***IDE de desarrollo***

Un Entorno de Desarrollo Integrado (Integrated Development Environment, IDE por sus siglas en inglés) consiste básicamente en un software que previamente ha sido instalado en la máquina, cuyo principal objetivo es el desarrollo de otro software. Es un entorno de programación que ha sido empaquetado como un programa de aplicación. Puede ser exclusivo para un lenguaje de programación o bien, para varios. Suele consistir de un editor de código (con facilidades como resaltado de sintaxis, completamiento de código y navegación entre clases), un compilador y herramientas de automatización de la compilación, un depurador y en algunos casos un constructor de interfaz gráfica. Una vez seleccionado el lenguaje para implementar y el tipo de aplicación a desarrollar, se valoró el uso de varios IDE como fueron: Aptana Studio, Eclipse, Spket y Netbeans. Para el desarrollo de nuestro sistema se asume el Netbeans, dada las características y comodidades que ofrece como herramienta para el desarrollo de la extensión para VP; por lo que se profundiza en el análisis de sus características.

**NetBeans 7.4**

El IDE NetBeans es un entorno integrado de desarrollo galardonado disponible para Windows, Mac, Linux y Solaris. NetBeans consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, escritorio y aplicaciones móviles utilizando la plataforma Java, así como JavaFX, PHP, JavaScript y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C/C++. Se escoge como IDE de desarrollo por ser el que propone la API de Visual Paradigm para el desarrollo de extensiones.

## 1.9 Conclusiones del capítulo

A lo largo de este capítulo se han abordado los diferentes elementos teóricos sobre los cuales se sustenta esta investigación que a través de su análisis permite llegar a las siguientes conclusiones parciales:

Se determinó la implementación de la extensión para la herramienta de modelado Visual Paradigm for UML en su versión 8.0, a partir de las necesidades del departamento Almacenes de Datos, ya que satisface las condiciones de desarrollo de la extensión y por estar contenido en el entorno tecnológico de DATEC.

Se identificaron los principales elementos a tener en cuenta para el desarrollo y modelado de los hechos, las dimensiones y las medidas a partir del análisis de las estructuras dimensionales.

Se escogió como metodología de desarrollo de software XP, teniendo en cuenta que es la que se adecua a las características del proyecto y al equipo de desarrollo.

Se empleó como IDE NetBeans 7.4 y Java como lenguaje de programación, ya que es el propuesto por la API de desarrollo de la herramienta VP.

# Capítulo II. Análisis y diseño de la solución para modelar soluciones de almacenes de datos.

## 2.1 Introducción

En la actualidad una de las actividades fundamentales en un proyecto es la planificación. Pues generalmente cuando se lleva a cabo el desarrollo de un proyecto, no se tienen totalmente claros sus aspectos y objetivos más importantes. De igual modo se desconoce el alcance que tendrá, tiempo de demora o esfuerzo a emplear para su construcción. En gran medida la planificación permite organizar de manera coherente un grupo de acciones encaminadas a lograr un producto final; con un elevado valor de uso, que satisfaga las necesidades del cliente y que esté dentro de los plazos de tiempo estimados (Mercado, 1995).

A partir de la metodología de desarrollo de software seleccionada en el capítulo anterior para la construcción de la extensión propuesta, se estudian 3 fases para llevar a cabo las tareas de planificación: la fase de exploración, la fase de planificación y la fase de iteración. Se presenta el modelo de dominio e historias de usuario, en las cuales se exponen las características de cada una de las funcionalidades y componentes que constituyen la extensión. Es definida además la arquitectura de la extensión, los patrones de diseño empleados y las tarjetas CRC identificadas para el desarrollo del presente trabajo de diploma.

## 2.2 Modelo de dominio de la extensión

Un modelo de dominio es un artefacto de la disciplina de análisis. Construido con las reglas de UML durante la fase de concepción. Contiene conceptos propios de la realidad física. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno del sistema. Muchos de los objetos o clases del dominio pueden obtenerse de la especificación de requisitos. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema. Este puede ser tomado como el punto de partida para el diseño del sistema. Un modelo de dominio es por tanto una representación de las cosas del mundo real del dominio de interés, muestra las clases conceptuales[6] o vocabulario del dominio.

[6] Informalmente, una clase conceptual es una idea, cosa u objeto. Más formalmente, una clase conceptual podría considerarse en términos de su símbolo, intensión, y extensión (Larman, 1999).

El siguiente diagrama es un modelo que muestra los conceptos fundamentales en el modelado de estructuras dimensionales. Como puede observarse, el número de conceptos es relativamente pequeño y debería resultar familiar a cualquiera que haya trabajado con AD. Se debe aclarar que este modelo de dominio es un diagrama con los objetos que existen (reales) en el diseño de un almacén y las relaciones que hay entre ellos, pero no se encuentran reflejadas clases de software, aunque en el futuro un elemento del modelo de dominio pueda pasar a ser una clase de software. En el siguiente modelo de dominio se representan los conceptos fundamentales (representados como entidades) de un AD: hecho y dimensión así como las relaciones entre ellos.

***Diagrama de dominio de la extensión***

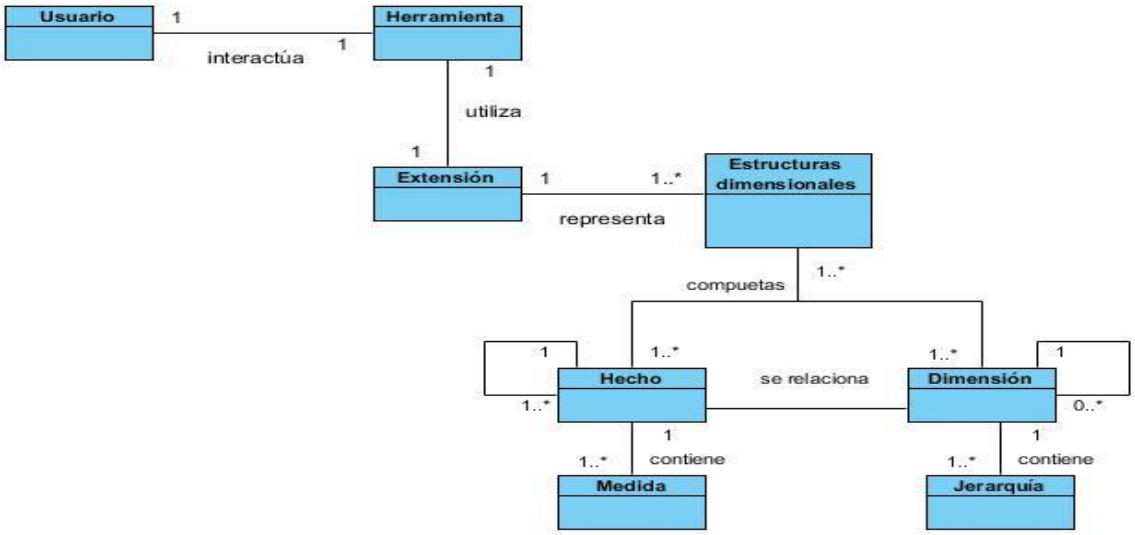


**Fig 8. Diagrama conceptual del dominio de la extensión**

**Descripción de los conceptos del diagrama de dominio de la extensión**

**Usuario:** persona facultada para utilizar la extensión asistida a la herramienta ¨Visual Paradigm for UML¨.

**Herramienta:** herramienta CASE ¨Visual Paradigm for UML¨ versión 1.0.

**Extensión:** componente integrado a la herramienta ¨Visual Paradigm for UML¨.

**Estructuras dimensionales:** buscan presentar los datos en un estándar y facilitar una recuperación adecuada de estos. Los datos son almacenados como hechos y dimensiones en un modelo de datos relacional.

**Hecho:** representa el concepto tal como se define en un almacén de datos. En ellos se representarán las medidas que generalmente son numéricas y que son la intersección de las dimensiones asociadas al hecho. Representa una tabla en el modelo físico.

**Medida:** datos numéricos que representan una actividad específica de un negocio.

**Dimensión:** característica de un hecho que permite su análisis posterior en el proceso de toma de decisiones y brinda una perspectiva adicional a un hecho dado. Son agrupaciones lógicas de atributos con un significado común y atómico. Representa una tabla en el modelo físico.

**Jerarquía:** en las jerarquías es donde se encuentran los atributos que se pueden agrupar por niveles, por ejemplo en la dimensión Tiempo, donde pueden existir los atributos Mes, Año y Día. Se les puede otorgar niveles a estos, dado que en un orden lógico Año puede estar en el primer nivel, Mes en el segundo y Día en el tercero, para buscar una cantidad de productos vendidos en el año Y, en el mes Z y en el día W.

## 2.3 Solución propuesta

Se propone realizar una extensión de la herramienta “Visual Paradigm for UML”, mediante un plugin que permita modelar físicamente el modelo dimensional de un AD. Con el objetivo de facilitar el diseño de modelos de datos dimensionales.

Para lograr un mejor resultado en el diseño de la extensión a desarrollar se hace necesario realizar una planificación de dicho proceso de desarrollo de software, basándose en las fases que propone la metodología seleccionada. Resultan de las mismas las tareas que reemplazarán un gran documento de requisitos.

**Pasos para desarrollar una extensión en la herramienta CASE “Visual Paradigm for UML”**

Para extender o crear una extensión en 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 sus funcionalidades y servicios mediante la librería openapi.jar. Esta se encuentra ubicada dentro de los paquetes de instalación de la herramienta, específicamente en el paquete “lib/openapi.jar”. Es necesario tener esta librería cargada en el proyecto.

**Implementación de plugin.xml y Plugin.java**

La clase Plugin.java permite mediante un script XML definido, la configuración de la extensión. Este será cargado por dicha clase 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 2 en el expediente de proyecto**.

**Ejemplo de implementación de la clase Plugin.java**

```
public class NamePlugin 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. Estas 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 3 en el expediente de proyecto**.

Mientras que el tercer paquete del proyecto contiene los diálogos que se necesitan mostrar a través del método performAction. Este está asociado a la acción correspondiente al diálogo 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 9 muestra la estructura de la plantilla de la extensión.


**Fig 9. Estructura de un proyecto de extensión para ¨Visual Paradigm for UML¨ en el IDE NetBeans.**

**Segundo paso:** integración de la extensión con la herramienta "Visual Paradigm for UML". Para dicha integración VP propone la siguiente estructura de despliegue de la extensión. Se crea una carpeta con el nombre de plugins dentro de la carpeta de instalación de VP la cual contiene la siguiente estructura de paquete:

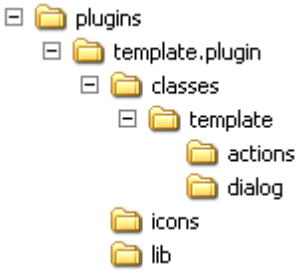


**Fig 10. Estructura de despliegue del plugin para ¨Visual Paradigm for UML¨.**

Si se observa dicha estructura de implementación de la extensión, difiere significativamente con respecto a la estructura de despliegue para la herramienta. Esto 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 11. 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

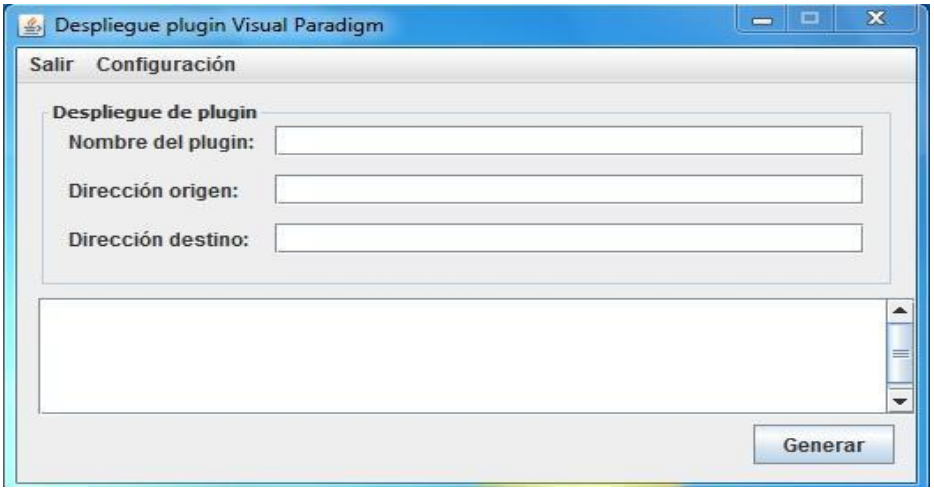


**Fig 11. Interfaz del despliegue de la extensión.**

Luego de detallar la estructura de desarrollo y los pasos para la integración del plugin, se pueden definir las fases planteadas por la metodología XP para el desarrollo del sistema, como se muestra a continuación.

## Fase de exploración

Esta es la primera fase que propone la metodología XP. En ella los clientes plantean a grandes rasgos las funcionalidades que son de interés para la elaboración del producto. Las mismas se transforman en HU. Partiendo de la información obtenida, el equipo de desarrollo se familiarizará con las herramientas, tecnologías y prácticas que serán utilizadas en el proyecto. Así como se explorarán las posibilidades de la arquitectura del sistema construyendo un prototipo para ello. La fase de exploración toma de pocas semanas a pocos meses, dependiendo de la habilidad que tengan los programadores con las tecnologías seleccionadas. Es necesario aclarar que las estimaciones llevadas a cabo en esta etapa son primarias, ya que las mismas se basan en datos de alto nivel los cuales pueden variar a medida que se analicen con mayor cuidado y detalle en las siguientes fases (Penadés, 2012).

***Historias de Usuario:*** "*Las Historias de Usuario son la técnica utilizada para especificar los requisitos del software*" (Canós, 2010). Estas describen las características y las funcionalidades requeridas para el software que se construirá (Pressman, 2002). Detallan cada uno de los requisitos del sistema, sean funcionales o no funcionales, sin hacer uso de extensos documentos. Además, su correcta definición será clave para guiar el proceso de implementación del software. Las 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 (Escribano, 2002).

Entre las principales características que debe tener una buena historia de usuario se encuentran: la historia debe ser entendida por el cliente (representa un concepto y no una especificación detallada). Cada historia debe devolver algún valor para el cliente, el tamaño de las historias de usuario debe ser tal que se puedan construir varias de ellas en una iteración (duración aproximada entre una-tres semanas ideales). Las historias deben ser independientes unas de otras, cada historia debe ser probable.

Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción, quizás además una estimación de esfuerzo en días. Otras de las actividades a tener en cuenta es la granularidad de las historias. Puede ser muy variable en dependencia de la complejidad del sistema, debido a esto, cada característica importante debe establecerse como una historia de usuario.

No hay que preocuparse si en un principio no se identifican todas las historias de usuario. Puesto que el proceso de elaboración de las iteraciones conlleva una retroalimentación en sí mismo. Al comienzo de cada iteración estarán registrados los cambios en las historias de usuario y es a partir de ello que se planificará la siguiente. Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración.

Como se planteó anteriormente una historia de usuario debe poder ser estimada, de lo contrario, si es muy grande, la misma es dividida en dos o más HU. El tiempo de estimación expuesto en las tablas es un aproximado, así como la prioridad de cada historia de usuario. En las historias de usuario más complejas se utiliza una programación exploratoria. Esta permite tener una idea del código a implementar para completar la misma. Es por ello que para la presente extensión son divididas algunas de las historias de usuario, y siguiendo los principios que conforman las buenas historias de usuario, quedan conformadas de la manera que se muestra en el presente trabajo de diploma.

Para el presente trabajo se realizaron un total de cinco HU. A continuación se muestra una de las HU definidas para la implementación de la extensión, las demás podrán ser consultadas en el expediente de proyecto.

**Tabla 2. H.U. Modelar físicamente el modelo dimensional de un AD.**

| **Historia de Usuario** | | |
|---|---|---|
| **ID:** 1 | **Nombre:** Modelar físicamente el modelo dimensional de un AD. | |
| **Descripción:** Esta HU permite insertar los modelos dimensionales en el diagrama físico y relacionar los mismos con los roles correspondientes para cada uno. Es descompuesta en las siguientes tareas de programación:<br>Permitir generar el script para cargar el diseño en el gestor de BD a partir del modelo físico.<br>Representar correctamente los hechos y sus elementos.<br>Representar correctamente las dimensiones y sus elementos.<br>Representar correctamente las relaciones entre los modelos dimensionales.<br>Representar correctamente el puente y sus elementos. | | |
| **Estimación aproximada:** 5 | **Prioridad:** Alta | **H.U original:** 1 |
| **Observaciones:** | | |

## Fase de planificación

El propósito general de la fase de planificación es que los clientes y desarrolladores se pongan de acuerdo en qué historias de usuario deben estar listas para la primera liberación. De forma particular, es en esta fase donde los programadores obtienen la estimación del esfuerzo necesario para la elaboración de las historias de usuario (Penadés, 2012). La medida para calcular dicho esfuerzo es el punto y por lo general, un punto equivale a una semana ideal. En el caso particular de la presente investigación se mantendrá durante todo el documento como valor de un punto, una semana ideal debido a las características del equipo de desarrollo y el entorno de trabajo.

El tiempo ideal es aquel en el cual se escribe el código sin distracciones y con una dedicación a tiempo completo. Esta estimación incluye todo el esfuerzo asociado a la implementación de la historia de usuario, como son: las pruebas unitarias, integración, refactorización del código, la preparación y ejecución de las pruebas de aceptación, entre otras actividades (Beck, 2009).

De acuerdo a los intereses del cliente será asignada la prioridad a cada historia de usuario. En vista a cumplir con uno de los principales objetivos que persigue XP: aumentar el valor del producto final en el menor tiempo de desarrollo posible. Es en esta fase donde se obtiene un aproximado del cronograma de entrega de cada una de las liberaciones. Esta es una fase de muy corta duración, pero clave en el avance de la extensión a desarrollar.

***Prioridad de las Historias de Usuario:*** la asignación de prioridad a las historia de usuario por el cliente es lo que decide el orden en el cual se implementarán las mismas. Siempre y cuando estén de acuerdo usuario y desarrollador. Las historias de usuario con menor número de prioridad son las más importantes, quedando conformadas de la siguiente manera:

**Tabla 3. Prioridad de las HU.**

| Historia de Usuario | Prioridad |
|---|---|
| Modelar físicamente el modelo dimensional de un AD | 1 |
| Permitir gestionar las especificaciones de los cubos, las medidas y las dimensiones con el mayor nivel de detalle posible. | 2 |
| Permitir administrar las jerarquías y sus niveles. | 3 |
| Generar y exportar un XML con el diseño de los cubos OLAP a partir del modelo físico. | 4 |
| Publicar el XML con el diseño de los cubos OLAP a partir del fichero y/o el diagrama en el Pentaho BI Server. | 5 |

***Estimación de esfuerzo de las Historias de Usuario:*** una vez asignadas las prioridades de las historias de usuario, los desarrolladores comienzan a estimar el esfuerzo necesario para la elaboración de cada una de ellas. Esta estimación es basada principalmente en la velocidad del equipo de desarrollo y en la semejanza con historias de usuario desarrolladas con anterioridad. Las historias de usuario de la presente investigación tienen un valor entre uno-tres puntos (medida utilizada normalmente, semanas ideales).

Ha de tenerse en cuenta que muy pocas veces este cronograma se lleva a cabo exactamente como se planifica. El esfuerzo necesario para construir las historias de usuario está basado en la técnica de estimación para el desarrollo ágil, quedando conformada de la siguiente manera:

**Tabla 4. Estimación del esfuerzo de las HU.**

| Historia de Usuario | Esfuerzo necesario (puntos estimados) |
|---|---|
| Modelar físicamente el modelo dimensional de un AD | 3 |
| Permitir gestionar las especificaciones de los cubos, las medidas y las dimensiones con el mayor nivel de detalle posible. | 1 |
| Permitir administrar las jerarquías y sus niveles. | 3 |
| Generar y exportar un XML con el diseño de los cubos OLAP a partir del modelo físico. | 3 |
| Publicar el XML con el diseño de los cubos OLAP a partir del fichero y/o el diagrama en el Pentaho BI Server. | 2 |

***Cronograma de liberación:*** cuando se planifica la liberación de un software se debe llevar un balance de la misma. Si se realiza muy pronto no se tendrán suficientes funcionalidades que ameriten dicha liberación. Por otro lado, esperar mucho tiempo entre liberaciones, llevará a que el software desarrollado quede atrás respecto a la competencia. Contrario a la creencia que existe sobre XP, que se debe llevar a cabo una liberación cada vez que es terminada una iteración, las liberaciones son lanzadas con un producto funcionando en óptimas condiciones (Beck, 2009). Ejemplo de un escenario donde no ocurre esto, es al momento de dividir historias de usuario. Esto llevaría a esperar varias iteraciones, de forma tal que al ser liberado éste, contenga todas las funcionalidades necesarias de la historia de usuario original. La planificación de la liberación es un esfuerzo unido entre el cliente y el desarrollador. El primero decide qué historias de usuario tienen la mayor prioridad y el segundo estima el tiempo que le llevará implementar las mismas (Beck, 2009).

Se debe tener en cuenta que en el cronograma de liberación pocas veces un plan marcha respecto a lo planificado. Es por esto que el plan de liberación está en constante cambio (Beck, 2009). Ejemplos de ello se pueden evidenciar, cada vez que el usuario decide cambiar la prioridad de una historia. Cuando el desarrollador divide o une historias de usuario, cuando surgen imprevistos en las tecnologías a utilizar, entre otras acciones. Es por ello que estos cambios deben ser aceptados y es necesario tenerlos en

cuenta. El cronograma de liberación de la presente investigación queda conformado de la siguiente manera:

**Tabla 5.Cronograma de liberación.**

| Iteración | Fecha de liberación |
|---|---|
| Primera iteración | 1ra semana de Abril de 2014 |
| Segunda iteración | |
| Tercera iteración | |
| Cuarta iteración | 2da semana de Mayo de 2014 |

***Fase de iteración***

Después que se describieron e identificaron las historias de usuario, así como estimado el esfuerzo que cada una de ellas posee, se procede a realizar la planificación de las etapas de implementación del sistema. Para ello se define un plan que contiene las iteraciones a realizar, así como cuáles historias de usuario serán implementadas y en qué orden para cada iteración, teniéndose en cuenta la duración de las mismas.

Destacar que en la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura. Sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio) (Penadés, 2012).
Las iteraciones quedan conformadas de la siguiente manera:

**Iteración 1:** en esta iteración se desarrolla la historia de usuario número uno. Esta permite modelar físicamente el modelo dimensional de un Almacén de Datos. Por la complejidad que presenta la misma es dividida en cinco tareas de programación, que son asignadas a los dos programadores del presente trabajo de diploma.

**Iteración 2:** en esta iteración se corregirán los errores o no conformidades del usuario identificados en la iteración anterior. Se desarrollará la historia de usuario número dos, la cual permitirá gestionar las especificaciones de los cubos, las medidas y las dimensiones con el mayor nivel de detalle posible. Por la

complejidad que presenta la misma, es dividida en tres tareas de programación que son asignadas a los dos programadores del presente trabajo de diploma.

**Iteración 3:** en esta iteración se corregirán los errores o no conformidades del usuario en la historia de usuario implementada en la iteración anterior. Se desarrollará la historia de usuario número tres. La misma permitirá administrar las jerarquías y sus niveles. Por la complejidad que presenta es dividida en cuatro tareas de programación que son asignadas a los dos programadores del presente trabajo de diploma.

**Iteración 4:** en esta iteración se corregirán los errores o no conformidades del usuario en la historia de usuario implementada en la iteración anterior. Se desarrollarán las historias de usuario número cuatro y número cinco. La HU 4 permitirá generar y exportar un archivo XML con el esquema de los cubos OLAP a partir del modelo físico. Mientras que la HU 5 permitirá una vez generado el XML con el diseño de los cubos, publicar el mismo en la herramienta "Pentaho BI Server" a partir del fichero y/o el diagrama con el diseño de los cubos. Además, se realizarán pruebas integrales a la extensión para validar todas sus funcionalidades corrigiéndose los errores o las no conformidades que se identifiquen y las que no han sido satisfechas en las iteraciones anteriores.

## 2.4 Arquitectura de software

Actualmente no existe una definición única de arquitectura de software. Varios autores han emitido sus criterios acerca del tema, con planteamientos diversos y con existencia de ideas comunes entre los mismos. Sin observarse planteamientos contradictorios, sino más bien complementarios. Vale destacar que todas giran en torno a tres ideas fundamentales: (1) Proceso dentro del ciclo de vida, (2) Topología, (3) Disciplina. Según Pressman "*La arquitectura de software de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen componentes del software, las propiedades visibles externamente de esos componentes y las relaciones entre ellos*" (Pressman, 2001).

Tal vez la definición mayoritariamente aceptada de arquitectura de software es la ofrecida por el estándar 1471 del Institute of Electrical and Electronics Engineers (IEEE, por sus siglas en inglés), arquitectura de software se entenderá como "*la organización fundamental de un sistema encarnada en sus componentes, las relaciones de los componentes con cada uno de los otros y con el entorno, y los principios que orientan su diseño y evolución*" (Kicillof, 2004).

Por lo que la arquitectura de software define el estilo o la combinación de estilos para una solución (para cumplir con los requisitos no funcionales), pues los requisitos funcionales se satisfacen mediante el modelado y diseño de la aplicación.

***Estilo de llamada y retorno***

El Estilo de llamada y retorno permite al diseñador de software construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se basa en la abstracción de procedimientos, funciones y métodos utilizados en grandes sistemas de software.

***Patrones***

Un patrón es una solución a un problema en un contexto. Codifica conocimiento específico acumulado por la experiencia en un dominio. Propone una forma de reutilizar la experiencia de los desarrolladores. Se basa en la recopilación del conocimiento de los expertos en desarrollo de software.

Un patrón en sentido general, define una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones. Estos pretenden: proporcionar catálogos de elementos reusables en el diseño de sistemas de software y evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente. Existen varios tipos de patrones, dependiendo del contexto particular en el cual se aplican o de la etapa en el proceso de desarrollo. Algunos de estos tipos son: patrones de diseño, de arquitectura, para ambientes distribuidos, entre otros. (Larman, 1999) En la presente investigación se abordan los patrones arquitectónicos y los patrones de diseño que son utilizados para conformar el diseño de la aplicación propuesta.

***Patrones arquitectónicos***

Los patrones arquitectónicos son patrones del software, que se encargan de definir la estructura de un sistema. Estos a su vez se componen de subsistemas con sus responsabilidades. Poseen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño. Un patrón arquitectónico se enfoca a dar solución a un problema en específico y abarca solo parte de la arquitectura. Aun cuando un patrón de este tipo brinda una imagen general de un sistema, él no es una arquitectura como tal. Es por esto que un patrón arquitectónico es un concepto que captura elementos esenciales de una arquitectura de software (Velásquez, 2009).

Los patrones arquitectónicos definen un enfoque específico para el manejo de algunas características de comportamiento del sistema (Pressman, 2001). Definen además la estructura básica de una aplicación. Proveen un subconjunto de subsistemas predefinidos, incluyendo reglas y pautas para su organización. Son una plantilla de construcción.

La herramienta “Visual Paradigm for UML”, provee una interfaz de programación de aplicaciones (API por sus siglas en inglés) que permite a los desarrolladores implementar y reutilizar clases e interfaces para

desarrollar funciones agregadas de software. Para definir la arquitectura de la extensión propuesta, resulta fundamental regirse por los elementos arquitectónicos definidos en el API de la herramienta. Esta propone una arquitectura basada en el patrón arquitectónico Modelo-Vista-Controlador (Model-View-Controller, MVC por sus siglas en inglés).

***Patrón Modelo-Vista-Controlador***

Este patrón arquitectónico pertenece a la familia de estilos de llamada y retorno que enfatizan la escalabilidad. Se basa en separar el modelo de datos de la aplicación de su representación de cara al usuario y de la interacción de éste con la aplicación, mediante la división de la aplicación en tres partes fundamentales:

- El modelo, que contiene la lógica de negocio de la aplicación.
- La vista, que muestra al usuario la información que éste necesita.
- El controlador, que recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización (Bahit, 2002).

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite por lo tanto construir y probar el modelo independientemente de la representación visual. Este patrón fue utilizado además de ser el propuesto por la API de la herramienta VP, ya que permite una separación de conceptos, de forma tal que el desarrollo de la aplicación esté estructurado de una mejor forma. También facilita la programación de manera paralela e independiente en las diferentes capas, además de brindarle una mayor escalabilidad.

***Patrones de diseño***

Los patrones de diseño son descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto (Gamma, 2008). Estos a su vez identifican las clases, instancias, roles, colaboraciones y la distribución de responsabilidades. Actualmente entre los más conocidos se encuentran los patrones de asignación de responsabilidades y los patrones de la banda de los cuatro.

**Patrones GRASP:** los Patrones Generales de Asignación de Responsabilidades de Software (General Responsibility Assignment Software Patterns, GRASP por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, de forma tal que se pueda diseñar software orientado a objetos (Grosso, 2011). Estos se conocen como: Experto, Creador, Bajo

Acoplamiento, Alta Cohesión, Controlador, Polimorfismo, Fábrica Pura, Indirección, y “No Hables con Extraños”, también conocido como “Variaciones Protegidas”.

**Experto:** el uso del patrón experto permite asignar a una clase la información necesaria para cumplir su responsabilidad. De esta forma, las funcionalidades son asignadas de forma adecuada, facilitando la futura reutilización de componentes. Uno de sus beneficios consiste en permitir conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar las acciones que se les solicita. La figura 12 muestra un ejemplo del empleo de éste patrón en el sistema.

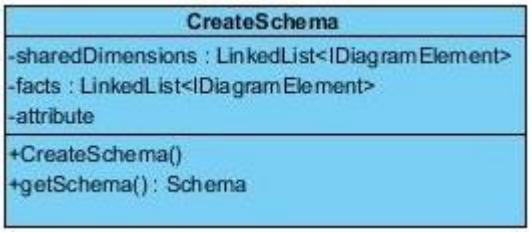


**Fig 12. Ejemplo del patrón experto.**

**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. En la Figura 12 anteriormente presentada se evidencia el empleo del patrón en el sistema.

**Alta cohesión:** este patrón permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. 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, de reutilizar, de mantener y delicadas, constantemente afectadas por los cambios.La figura 13 muestra un ejemplo del empleo de éste patrón en el sistema.

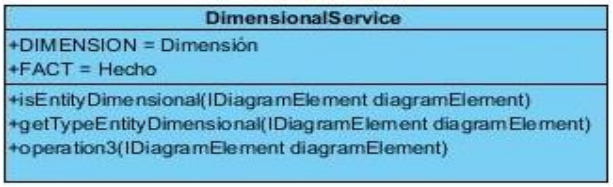


**Fig 13. Ejemplo del patrón alta cohesión.**

**Creador:** este patrón asigna responsabilidades relacionadas con la creación de objetos. Propone que un objeto tiene la responsabilidad de crear objetos de otra clase si agrega, contiene y usa exhaustivamente objetos de dicha clase. En la solución propuesta se utiliza al asignarle la responsabilidad a las controladoras de crear un objeto de las clases modelos para el posterior acceso a las funciones implementadas en el modelo. La figura 14 muestra un ejemplo del empleo de éste patrón en el sistema.

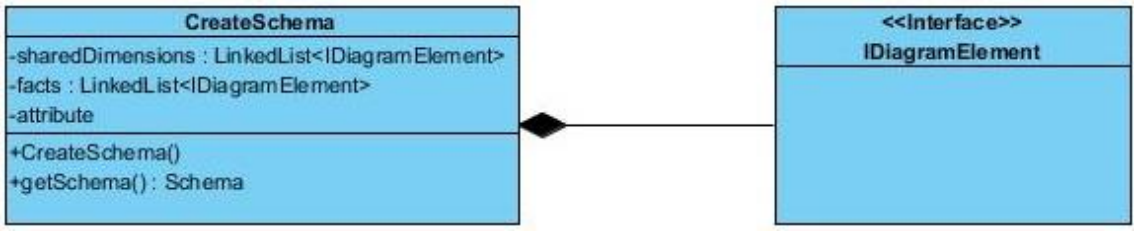


**Fig 14. Ejemplo del patrón creador.**

**Controlador:** es un patrón que sirve como intermediario entre una interfaz y el algoritmo que la implementa. De tal forma es la que recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. La figura 15 muestra un ejemplo del empleo de éste patrón en el sistema.

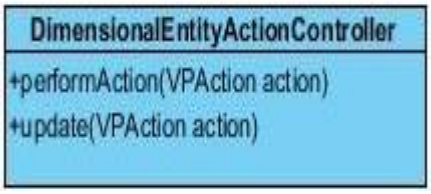


**Fig 15. Ejemplo del patrón controlador.**

**Patrones GoF:** los patrones de la Banda de los Cuatro (Gang of Four, GoF por sus siglas en inglés) son clasificados según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos.

**Según su propósito:**

1. Creacionales: resuelven problemas relativos a la creación de objetos.
2. Estructurales: resuelven problemas relativos a la composición de objetos.
3. De Composición: resuelven problemas relativos a la interacción entre objetos.

**Según su ámbito:**

1. Clases: relaciones estáticas entre clases.
2. Objetos: relaciones dinámicas entre objetos.

**Método de Fabricación (Factory Method):** para la implementación de extensiones en soluciones informáticas, se recomienda el uso de patrones como el conocido por Método de Fabricación. El cual se utiliza directamente en la implantación, garantizando la instanciación de los objetos. Dado que las extensiones en ocasiones son realizadas por externos al grupo de desarrollo que construyó el software para el cual se desea realizar la extensión. La figura 16 muestra un ejemplo del empleo de éste patrón en el sistema.

```
IDBTable table= IModelElementFactory.instance().createDBTable();
```

**Fig 16. Ejemplo del método de fabricación.**

**Iterador (Iterator)**

**Descripción:** el patrón iterador, es de tipo: comportamiento a nivel de objetos. Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. De esta forma se proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. Con la aplicación de este patrón se incrementa la flexibilidad, dado que para permitir nuevas formas de recorrer una estructura basta con modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. La figura 17 muestra un ejemplo del empleo de éste patrón en el sistema.

```
protected boolean isDimensionToFact(IDiagramElement DiagramElement) {

    if (getTypeEntityDimensional(DiagramElement).equals(DIMENSION)) {

        Iterator fromConnectors = DiagramElement.fromConnectorIterator();

        while (fromConnectors.hasNext()) {
            IConnectorUIModel connector = (IConnectorUIModel) fromConnectors.next();
            if (connector.getToShape() instanceof IDBTableUIModel
                    && getTypeEntityDimensional(connector.getToShape()).equals(FACT)) {
                return true;
            }
        }
    }
    return false;
}
```

**Fig 17. Ejemplo del método iterador.**

**Solitario (Singleton)**

**Descripción:** el patrón solitario, es de tipo comportamiento a nivel de objetos. Su propósito es garantizar la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. El acceso a la "instancia única" es controlado. Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario". Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable). La figura 18 muestra un ejemplo del empleo de éste patrón en el sistema.

```
IModelElementFactory modelElementFactory=IModelElementFactory.instance();

ApplicationManager applicationManager= ApplicationManager.instance()
```

**Fig 18. Ejemplo del método solitario.**

## 2.5 Tarjetas CRC

La metodología XP no recomienda la utilización de diagramas de clases, debido a la gran cantidad de documentación e inconvenientes que ello requiere, aunque tampoco excluye su elaboración. En su lugar orienta la realización de Tarjetas de Contenido, Responsabilidad, Colaboración, (CRC, por sus siglas en inglés), las cuales en cada una de las iteraciones van siendo definidas como herramientas de reflexión en el diseño de software orientado a objetos. Estas tarjetas brindan un acercamiento a la identificación de clases y la información referente a los objetos desarrollados en el proyecto. Cada tarjeta identifica una clase, sus responsabilidades y relaciones con otras clases. Una tarjeta CRC establece tres 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 dieciocho tarjetas CRC. A continuación se muestra una de las tarjetas CRC definidas para la extensión. El resto está ubicado en el expediente de proyecto de la solución.

**Tabla 6. Tarjeta CRC 1. DataWarehouseModeling.**

| **Clase: DataWarehouseModeling** | |
|---|---|
| **Responsabilidades:** | **Colaboraciones:** |
| 1. Permitir que VP reconozca la extensión.<br>2. Ejecutar en el inicio de la aplicación de VP los componentes para el correcto funcionamiento de la extensión.<br>3. Incluir una acción para adicionar un servicio a cada proyecto que se cree y se ejecutará sobre los diagramas entidad-relación.<br>4. Ejecutar acciones en el cierre de la aplicación. | ERDiagramRule |

## 2.6 Requisitos no funcionales

Al concebir las historias de usuario no se detallan ciertos aspectos que se deben precisar en la elaboración de una aplicación. Estos aspectos influyen en gran medida en el funcionamiento del producto final. La metodología XP no define artefactos para los requisitos no funcionales[7]. Sin embargo propone que los detalles de implementación de la historia de usuario se tienen en cuenta en el mismo momento de la concepción. Para un mejor funcionamiento es necesario que el producto cumpla con cualidades y propiedades; por estas razones se describen a continuación de forma general, algunas de las características no funcionales que debe poseer la extensión del presente trabajo de diploma.

[7] 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 RNF 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 (Sommerville, 2006).

**Requisitos de hardware**

La extensión requiere para su ejecución de una PC que debe cumplir como mínimo las siguientes especificaciones:

Procesador a 2.0 GHz o superior, 512 MB de RAM preferentemente 1GB o superior, 300 MB libres de disco duro.

**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.

**Requisitos de soporte**

La solución estará bien documentada de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarlo, permitiendo implementar cambios, ya sea de corrección, mejora o adaptación del software. Se proveerá un manual de usuario con todas las especificaciones de la extensión.

**Requisitos de portabilidad**

La extensión podrá ser integrada a VP en diferentes sistemas operativos, por ser VP multiplataforma.

**Apariencia o interfaz externa**

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

**Requisitos de software**

Para el uso de la extensión se deberá disponer del sistema operativo: Windows XP o superior, GNU/Linux preferentemente Ubuntu GNU/Linux 10.4 o superior, Debían 6.0 GNU/Linux o superior.

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

## 2.7 Conclusiones del capítulo

La elaboración del diagrama del dominio permitió un mejor entendimiento de la extensión, para así delimitar los elementos presentes y los que deberán ser implementados, con el objetivo de solucionar el problema de la investigación planteado.

La descripción de las consideraciones que se deben tener en cuenta para la implementación de extensiones para la herramienta Visual Paradigm for UML permitió un mayor entendimiento, facilitando el proceso de desarrollo e integración de la extensión.
La definición de las cinco historias de usuarios se realizó teniendo en cuenta las necesidades del cliente para la elaboración del modelo dimensional en las cuales quedaron descritos detalladamente los requisitos funcionales de la extensión.

La asignación de prioridades a las HU por parte del cliente y la estimación de esfuerzo necesario para su desarrollo por parte de los desarrolladores permitieron la confección del plan de iteraciones, en el que se evidencian las cuatro iteraciones definidas por el equipo de desarrollo para guiar la implementación de la extensión.

El empleo de patrones de diseño posibilitó una mejor asignación de responsabilidades entre las clases y estructurar correctamente los componentes a implementar, evidenciándose el uso de los patrones experto, bajo acoplamiento, alta cohesión, creador, controlador, método de fabricación, iterador y solitario.

La elaboración de las dieciséis tarjetas CRC, posibilitó la descripción de las clases de la extensión, todo lo cual tributa a una mejor comprensión de la solución y permite pasar a la siguiente fase del desarrollo.

# Capítulo III. Implementación y pruebas de la extensión para modelar soluciones de almacenes de datos.

## 3.1 Introducción

El presente capítulo se compone de dos acápites, Implementación y Pruebas. En la sección de implementación se definen las tareas a realizar una vez definidos los artefactos de análisis y diseño, así como las historias de usuario y las iteraciones. Por otra parte la sección de pruebas le permite a los desarrolladores, realizar ensayos constantemente en el producto buscando la mayor fiabilidad del mismo, intentando lograr la ausencia de errores en su desarrollo los cuales serán cada vez menores a medida que se avance en la codificación del mismo. Este capítulo tiene como objetivo evaluar y validar la calidad del sistema aplicando un conjunto de pruebas.

## 3.2 Implementación

Las tareas llevadas a cabo en esta etapa, ayudan a organizar la implementación y son solamente usadas por los programadores; pueden ser escritas utilizando un lenguaje técnico y no necesariamente deben ser entendibles por el cliente. Son asignadas a un equipo o programador responsable, aunque siempre la codificación se lleva a cabo por una pareja de programadores. Esta labor se efectúa para detallar mejor las historias de usuario, facilitando con ello el entendimiento en el proceso de implementación. Cada historia de usuario puede contener una o más tareas de programación, explicando de forma general las acciones que se realizan en la misma.

Acorde a la planificación realizada en el capítulo anterior, se llevaron a cabo cuatro iteraciones, donde se trabaja con cinco historias de usuarios que presentan un alto grado de complejidad, por lo que se descomponen en un total de dieciséis tareas de programación. Obteniéndose como finalidad un producto funcional con las características deseadas. A continuación se exponen ejemplos de algunas tareas de programación planificadas, para las historias de usuario uno y tres, correspondientes a las iteraciones número uno y número tres respectivamente. La demás tareas de programación se encuentran en el expediente de proyecto.

***Tareas de las historias de usuario***

**Iteración 1:** en esta iteración se desarrolló la historia de usuario número uno, para la cual se concibieron 5 tareas de programación. La Tabla 12 muestra la tarea número tres, la cual planifica la implementación de la funcionalidad que permite representar correctamente las dimensiones y sus elementos.

**Tabla 7. Tarea de Programación 3. Representar correctamente las dimensiones y sus elementos.**

| **Tareas de Historia de Usuario** | | |
|---|---|---|
| **ID:** 3 | **H.U:** Modelar físicamente el modelo dimensional de un AD. | **Iteración:**1 |
| **Nombre:** Representar correctamente las dimensiones y sus elementos. | | |
| **Tipo:** Desarrollo | **Puntos estimados:** 1 | |
| **Fecha inicio:** 4/02/14 | **Fecha Fin:** 10/02/14 | |
| **Responsable:** Alejandro Mariño Pérez – Ediel E García Amador. | | |
| **Descripción:** Se implementa una acción que permita generar una entidad con estereotipo dimensión e impide que a dicha entidad se le pueda eliminar el estereotipo dimensión. | | |

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

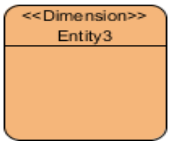


**Fig 19. Tarea de Programación 3. Representar correctamente las dimensiones y sus elementos.**

**Iteración 2:** en esta iteración se desarrolló la historia de usuario número dos, para la cual se concibieron 3 tareas de programación. La Tabla 13 muestra la tarea número siete, la cual planifica la implementación de la funcionalidad que permite gestionar correctamente las especificaciones de las dimensiones.

| Tabla 8. Tarea de Programación 7. Gestionar correctamente las especificaciones de las dimensiones. | | |
|---|---|---|
| **ID:** 7 | **H.U:** Permitir gestionar las especificaciones de los cubos, las medidas y las dimensiones con el mayor nivel de detalle posible. | **Iteración:**2 |
| **Nombre:** Gestionar correctamente las especificaciones las dimensiones. | | |
| **Tipo:** Desarrollo | **Puntos estimados:** 1 | |
| **Fecha inicio:** 14/03/14 | Fecha Fin: 16/03/14 | |
| **Responsable:** Alejandro Mariño Pérez – Ediel E García Amador. | | |
| **Descripción:** Se implementa el diseño de la interfaz que permite recoger las especificaciones de los cubos y se desarrolla una funcionalidad que permita hacer persistir los datos ingresados por el usuario en el hecho. | | |

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

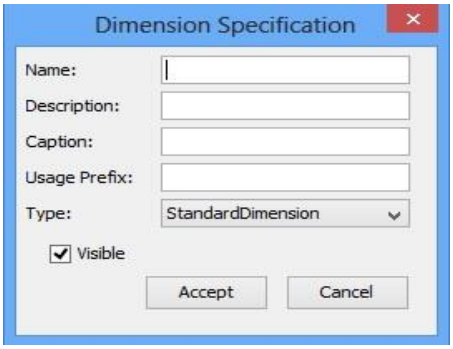


**Fig 20. Tarea de Programación 7. Gestionar correctamente las especificaciones de las dimensiones.**

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.

## 3.3 Estándar de codificación utilizado

Los estándares o convenciones de codificación que se emplean en el desarrollo de un software sobre la plataforma Java aglutinan un conjunto de técnicas de codificación sólidas y buenas prácticas de programación favoreciendo que se genere un código de programación de alta calidad. Este modelo de programación está basado en los estándares recomendados por Sun Microsystems, que han sido difundidos y aceptados ampliamente por toda la comunidad Java **(Flower, 2010)**.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento.

Estas normas son de gran importancia para la calidad del software, ya que presentan un marcado número de ventajas, entre las que se destacan:

- Facilitan el mantenimiento de una aplicación.
- Permiten que cualquier programador entienda y pueda mantener la aplicación.
- Mejoran la legibilidad del código, al mismo tiempo que permiten su compresión rápida.

En el desarrollo de la extensión para la herramienta Visual Paradigm for UML sobre la plataforma Java, se pusieron en práctica algunas de las convenciones recomendadas por Sun Microsystems. Entre ellas se pueden citar:

**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.

**Declaraciones:** toda variable local tendrá que ser inicializada en el momento de su declaración, a no ser que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

En las declaraciones de los métodos, no debe incluirse ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros. Los métodos se separarán entre sí mediante una línea en blanco.

**Sentencias:** el caracter inicio de bloque debe situarse al final de la línea que inicia el bloque. El caracter final de bloque debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer caracter de dicho bloque. Todas las sentencias de un bloque deben encerrarse entre llaves, aunque el bloque conste de una única sentencia.

**Espacios en blanco:** se utilizarán espacios en blanco entre una palabra clave y un paréntesis, tras cada coma en un listado de argumentos, para separar un operador binario de sus operandos, excepto en el caso del operador ("."), para separar las expresiones incluidas en la sentencia "for" y al realizar el moldeo o "casting" de clases.

**Nomenclatura de identificadores**

**Paquetes:** los nombres de los paquetes se escribirán siempre en letras minúsculas para evitar que entren en conflicto con los nombres de clases e interfaces.

**Clases e interfaces:** los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúscula. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúscula. Debe evitarse el uso de acrónimos o abreviaturas. Toda interfaz se nombrará con el prefijo "I" para diferenciarla de la clase que la implementa (que tendrá el mismo nombre sin el prefijo "I").

**Métodos:** los métodos deben ser verbos escritos en minúscula. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúscula.

**Variables:** las variables se escribirán siempre en minúscula. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúscula. Las variables nunca podrán comenzar con el caracter "_" o "$". Los nombres de variables deben ser cortos y debe evitarse el uso de nombres de variables con un solo caracter, excepto para variables temporales.

**Constantes:** todos los nombres de constantes tendrán que escribirse en mayúscula. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el caracter de subrayado "_".

**Buenas prácticas de programación**

**Visibilidad de atributos de instancia y de clase:** los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas; en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "get"[8] y "set"[9] correspondientes.

**Referencias a miembros de una clase:** debe evitarse el uso de objetos para acceder a los miembros de una clase (atributos y métodos estáticos). Debe utilizarse en su lugar el nombre de la clase.

**Asignación sobre variables:** se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, ya que dichas sentencias suelen ser difíciles de leer. No se deben utilizar asignaciones embebidas o anidadas.

**Paréntesis:** se deben utilizar paréntesis en expresiones que incluyan distintos tipos de operadores para evitar problemas de precedencia de operadores.

[8] Un método get obtiene el valor de un atributo específico.

[9] Un método set modifica o configura el valor de un atributo específico.

## 3.4 Pruebas de software

Las pruebas de software[10] son un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. Dichas pruebas son realizadas con el objetivo de detectar errores en el sistema, por lo que se llevan a cabo durante todo el ciclo de vida del producto. Los casos de prueba especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados **(Carreira, 2008)**.

En XP, uno de los pilares fundamentales es el proceso de pruebas, el cual anima a los desarrolladores a probar constantemente tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados. Este proceso es de vital importancia para el desarrollo, ya que permiten comprobar constantemente el código para así obtener un producto de mayor calidad. Es considerada una metodología "Test-drivenprogramming" (guiada por pruebas).

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.

La metodología XP propone el uso de dos tipos fundamentales de pruebas: unitarias y de aceptación; las primeras, confeccionadas y utilizadas por los desarrolladores para verificar el código, mientras que las últimas permiten al cliente constatar la correcta realización de las HU definidas.

***Pruebas unitarias***

Estas pruebas deben ser construidas antes que los métodos mismos, permitiéndole al programador tener máxima claridad sobre lo que va a programar antes de hacerlo, así como conocer cada uno de los casos de prueba que deberá pasar, lo que optimizará su trabajo y su código será de mejor calidad **(Tobón, 2007)**.

[10] Según la definición clásica de Myers "*Pruebas de software es el proceso de ejecución de un programa o sistema con la intención de encontrar errores*." (Myers, 1979).

Las pruebas unitarias son una forma de probar el buen funcionamiento de un módulo o una parte del sistema, con el fin de asegurar el correcto funcionamiento de todos los módulos por separado y evitar así errores futuros en el momento de la integración de todas sus partes.

Cabe destacar que una de las actividades que conforma la fase de pruebas en XP es la refactorización, constante actividad de reestructuración del código. Dicha actividad tiene como objetivo remover la duplicidad, mejorar la legibilidad, simplificar y hacer más flexible la codificación, para facilitar los posteriores cambios.

**Pruebas de aceptación**

Las pruebas de aceptación, también llamadas pruebas funcionales son supervisadas por el cliente basándose en los requerimientos tomados de las historias de usuario, su propósito es garantizar el cumplimiento de los requisitos pautados además de asegurar su correcto funcionamiento.

Las pruebas de aceptación son pruebas de caja negra, que representan un resultado esperado de determinada transacción con el sistema **(Tobón, 2007)**. Cada una de las pruebas representa una salida esperada del sistema, donde es responsabilidad del cliente verificar la corrección de las pruebas y tomar decisiones acerca de las mismas.

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 aceptación serán hechas por los asesores de calidad (clientes). Estas pruebas arrojarán los resultados que permitirán conocer el estado de la aplicación.

***Resultados de las pruebas hechas al sistema***

En la ejecución de las pruebas, adecuados 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:

**Pruebas unitarias**

Mediante las pruebas unitarias realizadas al código por los desarrolladores siguiendo la estrategia de **pruebas de unidad** concentrando el esfuerzo en la verificación del código implementado, se probaron

importantes caminos de control para descubrir errores dentro de los métodos implementados en la aplicación. Se examinaron las estructuras de datos para asegurar que mantienen la integridad durante todo el proceso de ejecución de los algoritmos. Detectándose un total de 38 NC, la clasificación y distribución de estas se muestra a continuación en la Figura 14.

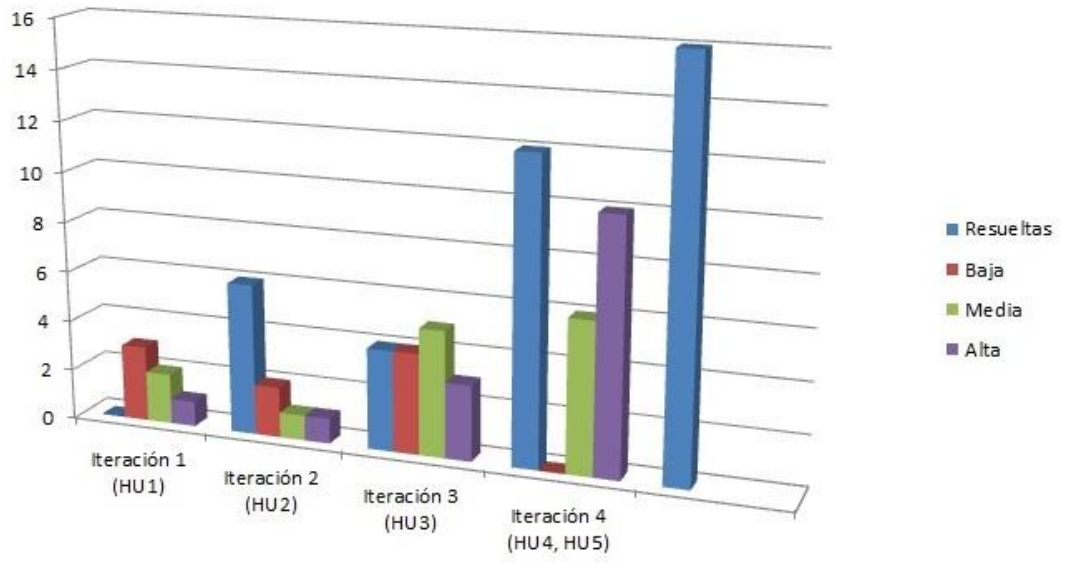


**Fig 21. Resultado de las pruebas unitarias realizadas al sistema.**

En respuesta a las NC 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, gramática o estandarización de los campos, los que se resolvieron de manera satisfactoria. Las NC detectadas fueron solucionadas al final de cada iteración, lo que constata que el producto está listo para su utilización.

**Pruebas de aceptación**

Las pruebas de aceptación fueron realizadas por el cliente, como propone la metodología bajo la cual se realiza el presente trabajo de diploma basándose en los requerimientos que se recogen en las historias de usuario, las cuales se realizaron por casos de pruebas, con el fin de validar que la aplicación desarrollada realice las funciones para las que ha sido creada en base a los requerimientos planteados por el usuario.

Inicialmente se propuso la realización de un caso de prueba por cada HU, por lo que se realizaron cinco casos de prueba.

Las pruebas de aceptación realizadas por los tutores, la oponente, los especialistas de calidad y los miembros de BI del departamento de almacenes de datos a la aplicación arrojaron un total de siete NC, las cuales se referían a errores de concordancia y uso indebido o excesivo de mayúsculas. A continuación se expone uno de los cinco casos de pruebas correspondiente a la HU ¨Modelar físicamente el modelo dimensional de un AD¨, los demás casos de prueba se encuentran en el expediente de proyecto de la metodología seleccionada. La tabla 11 evidencia el número de NC clasificadas en alta, media o baja, según su dificultad.

**Tabla 9. Prueba de Aceptación 1. Modelar estructuras dimensionales.**

| **Prueba de aceptación** | |
|---|---|
| **Código:** H.U1_P.A1 | **H.U:** Modelar físicamente el modelo dimensional de un AD. |
| **Nombre:** Modelar estructuras dimensionales. | |
| **Nombre de la persona que realiza la prueba:** Yaneisy Pedraza Gonzáles- José S Bermúdez. | |
| **Descripción de la prueba:** Probar que la aplicación permita modelar de forma correcta las estructuras dimensionales de un AD. | |
| **Condiciones de Ejecución:** El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada. | |
| **Entrada / Pasos de ejecución:** El usuario inserta las estructuras dimensionales en el diagrama entidad relación y las relaciona para obtener un modelo dimensional. | |
| **Resultado esperado:** Que la aplicación permita modelar de forma correcta todas las estructuras de datos dimensionales, con sus correspondientes atributos y relaciones. | |
| **Evaluación:** Satisfactoria | |

**Tabla 10. NC detectadas por las pruebas de aceptación**

| **No. NC** | **Alta** | **Media** | **Baja** | **Resueltas** |
|---|---|---|---|---|
| 7 | - | 3 | 4 | 7 |

En respuesta a las NC detectadas, se trabajó en la identificación de la dificultad y su ajuste o solución, según el caso correspondiente. Las NC detectadas fueron solucionadas correctamente, lo que constata que el producto está listo para su utilización.

## 3.5 Conclusiones del capítulo

La implementación de las dieciséis tareas de programación en las cuatro iteraciones realizadas permitió darle cumplimiento a todos los requerimientos funcionales, obteniendo de esta forma la extensión para modelar soluciones de almacenes de datos en la herramienta de modelado VP.

La realización de pruebas unitarias y de aceptación al producto, arrojaron como resultado un total de treinta y ocho no conformidades, las cuales fueron resueltas, permitiendo probar la calidad de las funcionalidades de la extensión, calificando la solución obtenida, como satisfactoria.

La realización de pruebas de liberación a la extensión por parte del grupo de aseguramiento de calidad y los especialistas de BI del departamento de almacenes de datos, arrojó dos no conformidades relacionadas con la estandarización del idioma las cuales se solucionaron, emitiéndose la carta de liberación del producto calificando la solución obtenida como satisfactoria.

## Conclusiones generales

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

Se identificaron los principales elementos a tener en cuenta para el desarrollo y modelado de los hechos, las dimensiones y las medidas a partir del análisis de las estructuras dimensionales.

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 7.4 y Java como lenguaje de programación, ya que es el propuesto por la API de desarrollo de la herramienta VP.

La elaboración del modelo de dominio permitió un mejor entendimiento de la extensión, para así delimitar los elementos presentes y los que fueron implementados, con el objetivo de solucionar el problema de la investigación planteado.

Para definir las características y condiciones que debe proveer la solución se identificaron cinco historias de usuarios, las cuales describen los requisitos funcionales de la extensión, además se elaboraron 18 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 de tareas de programación que permitieron el correcto desarrollo de la extensión, además se le realizaron las pruebas unitarias y de aceptación al producto, lográndose probar la calidad de las funcionalidades del mismo, calificando la solución obtenida como satisfactoria.

## Recomendaciones

A partir de las experiencias obtenidas en el desarrollo del trabajo de diploma y con el fin de adquirir un aprovechamiento óptimo del resultado alcanzado se recomienda:

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 funcionalidades que permitan cubrir todos los elementos que ofrece la herramienta Schema Workbench.

**Referencias Bibliográficas**

Bahit, Eugenia. 2002. El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC. s.l. : Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported, 2002.

Beck, Kent. 2009. Planning Extreme Programming. 2009. ISBN 0-201-7109-9.

Beltrá, Carlos Patricio López. 2007. Análisis, Diseño e implementación de un Data Mart para la Dirección Financiera y Recursos Humanos de la Escuela Politécnica del Ejército para una toma de decisión efectiva. 2007.

Bernabue, Ing. Ricardo Dario Córdoba. 2009. Data Warehousing: Investigación y Sistematización de Conceptos – Hefesto: Metodología propia para la Construcción de un Data Warehouse. Argentina : s.n., 2009.

Canós, José H. 2010. Métodologías Ágiles en el Desarrollo de Software. valencia : Universidad Politécnica de Valencia, 2010.

Carreira, Mercedes Ruiz. 2008. Estimación del Coste de la Calidad del Software a través de la Simulación de Procesos de Desarrollo. 2008.

Decsai. 2013. Modelado de datos. Fundamentos de diseño de bases de datos. [En línea] Departamento de Ciencias de la Computación. Universidad de Granada, 2013. [Citado el: 10 de Diciembre de 2013.] http://elvex.ugr.es/idbis/db/docs/intro/C%20Modelado%20de%20datos.pdf.

Escribano, Gerardo Fernández. 2002. Introducción a Extreme Programming. 2002.

Espinosa, Roberto. 2010. Pentaho Shema Worbench. [En línea] 2010. [Citado el: 29 de Octubre de 2013.] http://churriwifi.wordpress.com/2010/07/04/17-3-preparando-el-analisis-dimensional-definicion-de-cubos-utilizando-schema-workbench.

Fernández, Carlos. 2013. Dataprix. OLAP, MOLAP y ROLAP. [En línea] 2013. [Citado el: 1 de Diciembre de 2013.] http://www.dataprix.com/olap-rolap-molap.

Flower. 2010. Java Foundations. [En línea] 2 de Julio de 2010. [Citado el: 9 de Abril de 2014.] http://javafoundations.blogspot.com/2010/07/java-estandares-de-programacion.html.

Gallardo, Erith Eduardo Pérez. 2012. Data Warehouse, Modelo, Conceptos e Implementación orientada a SQL Server. [En línea] 2012. [Citado el: 15 de Noviembre de 2013.] http://www.monografias.com/trabajos57/data-warehouse-sql/data-warehouse-sql2.shtml.

Gamma, Erick. 2008. Patrones de Diseño: elementos de software orientados a objetos reutilizables. Traducido por: Acebal, C. F. Addison Wesley, 313 p. Traducido de: Design Patterns: Elements of Reusable Object-Oriented Software. 2008.

Garlan, M SHAW - D. 1996. Software Architecture Perspective on an Emerging Discipline. New Jersey : Prentice Hall, 1996.

Grosso, Andrés. 2011. Prácticas de Software: Patrones Grasp. [En línea] 2011. [Citado el: 17 de Marzo de 2014.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

Guerrero. 2011. El rol del Analista en RUP 2007. [En línea] 2011. [Citado el: 4 de Diciembre de 2013.] http://hanzcocchi.net/el-rol-del-analista-en-rup.

Hernán. 2009. Diseño de una metodología Ágil para el desarrollo de software. 2009.

Herrera, Cristhian. 2007. Adictos al trabajo. [En línea] 2007. [Citado el: 10 de Noviembre de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=datawarehouse4.

IBM. 2013. Creación de modelos dimensionales lógicos y físicos. [En línea] 2013. [Citado el: 16 de Noviembre de 2013.] http://pic.dhe.ibm.com/infocenter/idm/docv3/index.jsp?topic=%2Fcom.ibm.datatools.dimensional.ui.doc%2Ftopics%2Ft_ldm_pdm_dim_not_wizard.html.

Inmon, William H. 2005. Building the Data Warehouse. Fourth Edition. Indianapolis : s.n., 2005. ISBN-10: 0-7645-9944-5.

Inmon, Willian Bill. 2002. Building Data Warehouse. 2002.

Jacobson, James. 2010. El Proceso Unificado de Desarrollo de Software. 2010.

Jeffries, Ron. 2000. Extreme Programming Installed. Addison-Wesley Professional. 2000. ISBN 0-201-70842-6.

Kent Beck, Addison-Wesley y otros. Planning Extreme Programming. 2000. 160 p. ISBN 0-201-7109-9.

Kicillof, Carlos Reynoso – Nicolás. 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Universidad de Buenos Aires : s.n., 2004.

Kimball. 1998. The data Warehouse LifecylecToolkit. New York : s.n., 1998.

Kimball, Ralph. 2002. The Data Warehouse ETL Toolkit. Second Edition. 2002.

Larman, Craig. 1999. UML y Patrones. Una Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado, 2da Edición. 1999. ISBN 84-205-3438-2.

León, Rolando Alfredo Hernández. 2002. El paradigma cuantitativo de la investigación científica. Ciudad de la Habana : Editorial Universitaria EDUNIV, 2002. ISBN: 959-16-0343-6.

López, Pedro. 2012. [En línea] 2012. [Citado el: 17 de Diciembre de 2013.] http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.

Marches, Michele. 2002. Extreme Programming Perspectivas. Addison Wesley. 2002. ISBN 0-201-77005-9.

Medina, Cdte. Gustavo Moya Pérez - Darisleidy Arcia. 2012. Modelación, Implementación e Integración de la Herramienta de Gestión del XML para Almacenes de Datos. La Habana : s.n., 2012.

Mercado, Salvador. 1995. Administración Aplicada. Teoría y Práctica. s.l. : Editoral Limusa, 2002, 1995. ISBN.

Myers, G.J. 1979. The Art of Software Testing. 1979.

Ortiz, Antonio Moreno. 2010. [En línea] Facultad de Filosofía y Letras. Universidad de Málaga, 2010. [Citado el: 12 de Diciembre de 2013.] http://elies.rediris.es/elies9/4-2.htm.. ISSN: 1139-8736 .

paradigm, Visual. 2010. UML and Business Process modeling tool for software development project - Visual Paradigm for UML. [En línea] 2010. [Citado el: 20 de Noviembre de 2013.] http://www.visual-paradigm.com/product/vpuml/index.jsp.

Penadés, Patricio Letelier- María Carmen. 2012. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 2012. [Citado el: 3 de Marzo de 2014.] http://www.cyta.com.ar/ta0502/v5n2a1.htm.

Pressman. 2002. Desarrollo Ágil. Capítulo 4. 6ta edición. 2002.

Pressman, Roger S. 2001. Ingeniería de Software. 6ta Edición. Capítulo 10: Diseño arquitectónico. Madrid. España : Editorial Mac Graw Hill, 2001.

Salomón, Omar Pérez. 2014. [En línea] 2014. [Citado el: 4 de Diciembre de 2013.] http://www.ecured.cu/index.php/Historia_de_la_Inform%C3%A1tica_en_Cuba#La_UCI.

Sampieri, Roberto Hernández. 1998. Metodología de la investigación. Segunda edición. México : s.n., 1998. ISBN 970-10-1899-0.

Sanchez, María A. Mendoza. 2004. Metodologías De Desarrollo De Software. 2004.

Sinnexus. 2012. [En línea] 2012. [Citado el: 28 de Noviembre de 2013.] http://www.sinnexus.com/business_intelligence/olap_avanzado.aspx.

Sommerville. 2006. Software Engineering. 2006. ISBN 0-321-31379-8.

Suárez, Sergio Martin Torres-Alain Suárez. 2012. Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado “Visual Paradigm for UML”. La Habana : s.n., 2012.

Summan. 2014. [En línea] 2014. [Citado el: 4 de Abril de 2014.] http://www.summan.com/pentaho/pentaho-bi-platform-server.

Thomsen, Erick. 2002. OLAP Solutions, Building Multidimensional Information Systems ,Second Edition. Inc. USA : Jhon Wiley & Sons : s.n., 2002.

Tobón, Luis Miguel Echeverry. 2007. Caso práctico de la metodología ágil XP al desarrollo de software. 2007.

Urquizu, Pau. 2012. [En línea] 2012. [Citado el: 19 de Diciembre de 2013.] http://www.businessintelligence.info/definiciones/que-es-modelo-dimensional.html.

Vazquez, Ing. Yanet Peña. 2010. Técnicas y herramientas de inteligencia de negocios para el análisis de datos. 2010.

Velásquez, Key. 2009. Patrones Arquitectónicos. [En línea] 2009. [Citado el: 20 de Noviembre de 2014.] http://www.buenastareas.com/ensayos/Patrones-Arquitectonicos/1069013.html.

Zorrilla, Marta. 2008. Data warehouse y OLAP. Universidad de Cantabria : s.n., 2008.

**Bibliografía**


1. Bahit, Eugenia. 2002. El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC. s.l. : Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported, 2002.
2. Beck, Kent. 2009. Planning Extreme Programming. 2009. ISBN 0-201-7109-9.
3. Beltrá, Carlos Patricio López. 2007. Análisis, Diseño e implementación de un Data Mart para la Dirección Financiera y Recursos Humanos de la Escuela Politécnica del Ejército para una toma de decisión efectiva. 2007.
4. Bernabue, Ing. Ricardo Dario Córdoba. 2009. Data Warehousing: Investigación y Sistematización de Conceptos – Hefesto: Metodología propia para la Construcción de un Data Warehouse. Argentina : s.n., 2009.
5. Canós, José H. 2010. Métodologías Ágiles en el Desarrollo de Software. valencia : Universidad Politécnica de Valencia, 2010.
6. Carreira, Mercedes Ruiz. 2008. Estimación del Coste de la Calidad del Software a través de la Simulación de Procesos de Desarrollo. 2008.
7. Decsai. 2013. Modelado de datos. Fundamentos de diseño de bases de datos. [En línea] Departamento de Ciencias de la Computación. Universidad de Granada, 2013. [Citado el: 10 de Diciembre de 2013.] http://elvex.ugr.es/idbis/db/docs/intro/C%20Modelado%20de%20datos.pdf.
8. Escribano, Gerardo Fernández. 2002. Introducción a Extreme Programming. 2002.
9. Espinosa, Roberto. 2010. Pentaho Shema Worbench. [En línea] 2010. [Citado el: 29 de Octubre de 2013.] http://churriwifi.wordpress.com/2010/07/04/17-3-preparando-el-analisis-dimensional-definicion-de-cubos-utilizando-schema-workbench.
10. Fernández, Carlos. 2013. Dataprix. OLAP, MOLAP y ROLAP. [En línea] 2013. [Citado el: 1 de Diciembre de 2013.] http://www.dataprix.com/olap-rolap-molap.
11. Flower. 2010. Java Foundations. [En línea] 2 de Julio de 2010. [Citado el: 9 de Abril de 2014.] http://javafoundations.blogspot.com/2010/07/java-estandares-de-programacion.html.
12. Gallardo, Erith Eduardo Pérez. 2012. Data Warehouse, Modelo, Conceptos e Implementación orientada a SQL Server. [En línea] 2012. [Citado el: 15 de Noviembre de 2013.] http://www.monografias.com/trabajos57/data-warehouse-sql/data-warehouse-sql2.shtml.

13. Gamma, Erick. 2008. Patrones de Diseño: elementos de software orientados a objetos reutilizables. Traducido por: Acebal, C. F. Addison Wesley, 313 p. Traducido de: Design Patterns: Elements of Reusable Object-Oriented Software. 2008.
14. Garlan, M SHAW - D. 1996. Software Architecture Perspective on an Emerging Discipline. New Jersey : Prentice Hall, 1996.
15. Grosso, Andrés. 2011. Prácticas de Software: Patrones Grasp. [En línea] 2011. [Citado el: 17 de Marzo de 2014.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.
16. Guerrero. 2011. El rol del Analista en RUP 2007. [En línea] 2011. [Citado el: 4 de Diciembre de 2013.] http://hanzcocchi.net/el-rol-del-analista-en-rup.
17. Hernán. 2009. Diseño de una metodología Ágil para el desarrollo de software. 2009.
18. Herrera, Cristhian. 2007. Adictos al trabajo. [En línea] 2007. [Citado el: 10 de Noviembre de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=datawarehouse4.
19. IBM. 2013. Creación de modelos dimensionales lógicos y físicos. [En línea] 2013. [Citado el: 16 de Noviembre de 2013.] http://pic.dhe.ibm.com/infocenter/idm/docv3/index.jsp?topic=%2Fcom.ibm.datatools.dimensional.ui.doc%2Ftopics%2Ft_ldm_pdm_dim_not_wizard.html.
20. Inmon, William H. 2005. Building the Data Warehouse. Fourth Edition. Indianapolis : s.n., 2005. ISBN-10: 0-7645-9944-5.
21. Inmon, Willian Bill. 2002. Building Data Warehouse. 2002.
22. Jacobson, James. 2010. El Proceso Unificado de Desarrollo de Software. 2010.
23. Jeffries, Ron. 2000. Extreme Programming Installed. Addison-Wesley Professional. 2000. ISBN 0-201-70842-6.
24. Kent Beck, Addison-Wesley y otros. Planning Extreme Programming. 2000. 160 p. ISBN 0-201-7109-9.
25. Kicillof, Carlos Reynoso – Nicolás. 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Universidad de Buenos Aires : s.n., 2004.
26. Kimball. 1998. The data Warehouse LifecylecToolkit. New York : s.n., 1998.
27. Kimball, Ralph. 2002. The Data Warehouse ETL Toolkit. Second Edition. 2002.
28. Larman, Craig. 1999. UML y Patrones. Una Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado, 2da Edición. 1999. ISBN 84-205-3438-2.

29. León, Rolando Alfredo Hernández. 2002. El paradigma cuantitativo de la investigación científica. Ciudad de la Habana : Editorial Universitaria EDUNIV, 2002. ISBN: 959-16-0343-6.
30. López, Pedro. 2012. [En línea] 2012. [Citado el: 17 de Diciembre de 2013.] http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.
31. Marches, Michele. 2002. Extreme Programming Perspectivas. Addison Wesley. 2002. ISBN 0-201-77005-9..
32. Medina, Cdte. Gustavo Moya Pérez - Darisleidy Arcia. 2012. Modelación, Implementación e Integración de la Herramienta de Gestión del XML para Almacenes de Datos. La Habana : s.n., 2012.
33. Mercado, Salvador. 1995. Administración Aplicada. Teoría y Práctica. s.l. : Editoral Limusa, 2002, 1995. ISBN.
34. Myers, G.J. 1979. The Art of Software Testing. 1979.
35. Ortiz, Antonio Moreno. 2010. [En línea] Facultad de Filosofía y Letras. Universidad de Málaga, 2010. [Citado el: 12 de Diciembre de 2013.] http://elies.rediris.es/elies9/4-2.htm.. ISSN: 1139-8736 .
36. paradigm, Visual. 2010. UML and Business Process modeling tool for software development project -Visual Paradigm for UML. [En línea] 2010. [Citado el: 20 de Noviembre de 2013.] http://www.visual-paradigm.com/product/vpuml/index.jsp.
37. Penadés, Patricio Letelier- María Carmen. 2012. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 2012. [Citado el: 3 de Marzo de 2014.] http://www.cyta.com.ar/ta0502/v5n2a1.htm.
38. Pressman. 2002. Desarrollo Ágil. Capítulo 4. 6ta edición. 2002.
39. Pressman, Roger S. 2001. Ingeniería de Software. 6ta Edición. Capítulo 10: Diseño arquitectónico. Madrid. España : Editorial Mac Graw Hill, 2001.
40. Salomón, Omar Pérez. 2014. [En línea] 2014. [Citado el: 4 de Diciembre de 2013.] http://www.ecured.cu/index.php/Historia_de_la_Inform%C3%A1tica_en_Cuba#La_UCI.
41. Sampieri, Roberto Hernández. 1998. Metodología de la investigación. Segunda edición. México : s.n., 1998. ISBN 970-10-1899-0.
42. Sanchez, María A. Mendoza. 2004. Metodologías De Desarrollo De Software. 2004.
43. Sinnexus. 2012. [En línea] 2012. [Citado el: 28 de Noviembre de 2013.] http://www.sinnexus.com/business_intelligence/olap_avanzado.aspx.
44. Sommerville. 2006. Software Engineering. 2006. ISBN 0-321-31379-8.

45. Suárez, Sergio Martin Torres-Alain Suárez. 2012. Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado “Visual Paradigm for UML”. La Habana : s.n., 2012.
46. Summan. 2014. [En línea] 2014. [Citado el: 4 de Abril de 2014.] http://www.summan.com/pentaho/pentaho-bi-platform-server.
47. Thomsen, Erick. 2002. OLAP Solutions, Building Multidimensional Information Systems ,Second Edition. Inc. USA : Jhon Wiley & Sons : s.n., 2002.
48. Tobón, Luis Miguel Echeverry. 2007. Caso práctico de la metodología ágil XP al desarrollo de software. 2007.
49. Urquizu, Pau. 2012. [En línea] 2012. [Citado el: 19 de Diciembre de 2013.] http://www.businessintelligence.info/definiciones/que-es-modelo-dimensional.html.
50. Vazquez, Ing. Yanet Peña. 2010. Técnicas y herramientas de inteligencia de negocios para el análisis de datos. 2010.
51. Velásquez, Key. 2009. Patrones Arquitectónicos. [En línea] 2009. [Citado el: 20 de Noviembre de 2014.] http://www.buenastareas.com/ensayos/Patrones-Arquitectonicos/1069013.html.
52. Zorrilla, Marta. 2008. Data warehouse y OLAP. Universidad de Cantabria : s.n., 2008.