*Universidad de las Ciencias Informáticas*

*Facultad 6*


# *Título: Componente del Pentaho Data Integration para el proceso de perfilado de datos.*

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

***Autores:***
*Mauricio Moreno Molina*
*Diana Rosa Boggiano González*

***Tutores:***
*Ing.Yosbel Rodríguez Rodríguez*
*Ing. Wendy Romalde Ruiz*

*La Habana, junio, 2014*

*"Año 56 de la Revolución"*

*La Educación es el pasaporte hacia el futuro, el mañana pertenece a aquellos que se preparan para él en el día de hoy.*

Malcolm X

# *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 firmo la presente a los ____ días del mes de _____ del año _____.

**<nombre autor>**

_____

Firma del Autor

**<nombre tutor>**

_____

Firma del Tutor

# Datos de contacto

**Autores:**

Mauricio Moreno Molina
Universidad de las Ciencias Informáticas,
La Habana, Cuba
**E-mail:** mmoreno@estudiantes.uci.cu

Diana Rosa Boggiano González
Universidad de las Ciencias Informáticas,
La Habana, Cuba
**E-mail:** drboggiano@estudiantes.uci.cu

**Tutor:**

**Ing.** Yosbel Rodríguez Rodríguez
Universidad de las Ciencias Informáticas,
La Habana, Cuba
**E-mail:** yrdguezro@uci.cu

**Cotutor:**

**Ing.** Wendy Romalde Ruiz
Universidad de las Ciencias Informáticas,
La Habana, Cuba
**E-mail:** wromalde@uci.cu

***Diana:***

*Quisiera agradecer en primer lugar a mis padres* ***Dianelis*** *y* ***David*** *por haberme dado la vida y por haberme apoyado en todas mis decisiones.*

*A mis abuelos por haber estado junto a mí toda mi vida y por darme una buena educación, gracias mima y papá.*

*A mis tías* ***Cari*** *y* ***Marlen*** *por ser las mejores amigas, por apoyarme, aconsejarme y brindarme mucho cariño.*

*A mi hermanita* ***Angélica*** *por permitirme ser un ejemplo para ella.*

*A todos mis primos:* ***Sam****,* ***Dariel****,* ***Yasmany*** *y* ***Yulier****.*

*En especial a mi tía* ***Taimara*** *por haber sido una madre para mí, aunque este lejos ha sido mi ejemplo a seguir y gracias a ella estoy aquí.*

*A todas las personas que de una forma u otra me apoyaron en el transcurso de la carrera, a* ***Katy****,* ***Ariannis****,* ***Annia****,* ***Angel****,* ***Oscar*** *y* ***Deimar*** *por ser excelentes personas y buenos amigos.*

*A mi compañero de tesis* ***Mauricio*** *por ser paciente y comprensivo y por apoyarme durante todo este tiempo.*

*A* ***Javier*** *muchas gracias por estar junto a mí en casi toda la carrera y por haberme brindado momentos maravillosos que nunca olvidaré.*

*A mis tutores* ***Wendy*** *y* ***Yosbel*** *por el apoyo depositado en el desarrollo de este trabajo.*

***Mauricio:***

*A mi mamá* ***Thelma*** *gracias por permitirme conocer la vida, enseñarme a andar, por ser la voz que me impulsa y me apoya constantemente en todas las metas que me propongo. Gracias por enseñarme que aunque el trayecto parezca difícil siempre, a partir del esfuerzo, se puede salir adelante. A ti te lo debo todo y nunca será suficiente.*

*A mi papá* ***Leonardo*** *porque desde pequeño me has apoyado en todo lo que ha estado a tu alcance, la voz fuerte que me enseñó que tengo que seguir hacia adelante, no importan los obstáculos y que todo está en el empeño que le ponga a las tareas.*

*A mis abuelos* ***Caridad****,* ***Delio*** *y* ***Ubaldo****. Gracias por aconsejarme y confiar en mí siempre.*

*Al resto de mi extensa familia que de una forma u otra han brindado su grano de arena para ayudarme a construir este sueño.*

*A* ***Idalberto*** *y* ***Miriam*** *que son mis abuelos adoptivos y siempre me han apoyado en mis metas, por ser las dos personas que más orgullo sienten cuando hablan de mí. Gracias de todo corazón.*

*A mis tutores por confiar en nosotros para desarrollar esta investigación, que en un principio parecía una misión imposible, pero con su ayuda, apoyo y experiencia, poco a poco salieron los resultados.*

*A todos mis amigos, los que me han aguantado durante los 5 años, así como los que se fueron uniendo durante el transcurso de esta travesía y a los que ya no están. Gracias a todos los que de una forma u otra han ayudado a mi formación, a todos los que me han ayudado a soportar la lejanía, a todos los que se convirtieron en mi familia. En especial a* ***Yusniel****,* ***Alián****,* ***Jorge Luis****,* ***Yadián****,* ***Jaciel****,* ***Luisma****,* ***Alejandro*** *que más que amigos fueron hermanos. A mí hijo adoptivo* ***Deimar****. En fin, gracias a todos por acogerme y hacerme parte de sus vidas.*

*A mi compañera de tesis* ***Diana Rosa*** *por la preocupación y el apoyo durante todo este tiempo. Gracias por comprenderme y confiar en mí.*

***Diana:***

*Dedico este trabajo de diploma a mi mamá y a mi papá; a mis abuelos porque siempre confiaron en mí y a toda mi familia que de una forma u otra siempre me brindaron su apoyo incondicional.*

***Mauricio:***

*Dedico este trabajo de diploma a mi abuela Eulalia por dedicarse en tiempo y alma a mi crianza y contribuir de manera decisiva en mi formación y a convertirme en la persona que soy. A mis padres y amigos.*

**RESUMEN**

En las últimas décadas se ha evidenciado un crecimiento exponencial de la información debido al constante desarrollo de la tecnología y la sociedad a pasos agigantados. Realizar la gestión del gran cúmulo de datos que se generaban diariamente en múltiples empresas comenzó a tornarse un proceso complejo debido a la variedad de las fuentes de los datos y a la falta de una normalización en los mismos.

En la investigación realizada se incluyen diversos aspectos relacionados con la plataforma Pentaho y específicamente con su herramienta Pentaho Data Integration (PDI), así como del proceso de perfilado de datos que se realiza a partir de la misma. Los beneficios de este proceso y la necesidad de actualización del mismo para el óptimo desarrollo de la gestión de los datos, impulsó al desarrollo de un componente que facilita y apoya el trabajo de perfilado de datos con el PDI.

El ciclo de desarrollo del componente para el PDI posibilita documentar una serie de resultados que permitirán una futura profundización sobre el tema, así como la comparación con otras tecnologías y la sustitución de las mismas. Se describen las características de la aplicación y las actividades relacionadas con la metodología adoptada durante todo el proceso.

PALABRAS CLAVE: componente, perfilado de datos, Pentaho Data Integration, desarrollo.

**Abstract**

In recent decades it has shown an exponential growth of information due to the constant development of technology and society rapidly. Perform management of the large mass of data being generated daily in multiple companies began to turn a complex process because of the variety of data sources and lack of standardization in them.

Various aspects of the Pentaho platform and specifically his tool Pentaho Data Integration (PDI) and the data profiling process that is performed after the same are included in the investigation. The benefits of this process and the need to update the same for the optimal development of data management, prompted the development of a component that facilitates and supports the work of profiling data with the PDI.

The development cycle for the PDI component enables document a series of results that will enable future deepening on the subject, as well as comparison with other technologies and replace them. The characteristics of the application and activities related to the methodology adopted throughout the process are described. KEYWORDS: component, data profiling, Pentaho Data Integration, development.

## Contenido

Introducción ..... 1

**CAPÍTULO1: FUNDAMENTOS TEÓRICOS.** ..... 5

1.1 Conceptos básicos asociados al problema. ..... 5

1.2 Perfilado de datos ..... 6

1.3 Herramientas existentes para perfilado de datos. ..... 9

1.4 Pentaho Data Integration ..... 11

1.5 Metodología de desarrollo ..... 15

1.6 Herramientas y tecnologías utilizadas ..... 20

*1.6.1 Herramienta CASE* (Ingeniería de Software Asistida por Computadora) ..... 20

*1.6.2 Lenguaje de programación* ..... 21

*1.6.3 Entorno de desarrollo integrado* ..... 22

Conclusiones parciales ..... 24

**CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA.** ..... 25

2.1 Modelo del dominio ..... 25

*2.1.1 Descripción de las clases del dominio* ..... 25

2.2 Propuesta de solución ..... 26

2.3 Especificación de los requisitos ..... 26

*2.3.1 Requisitos funcionales* ..... 26

2.4 Definición de los casos de uso ..... 29

*2.4.1 Descripción de los actores del sistema* ..... 29

*2.4.2 Descripción del caso de uso Realizar análisis general en un flujo de datos.* ..... 31

2.5 Diagramas de clases de diseño ..... 33
2.6 Patrones utilizados ..... 35
2.7 Patrón arquitectónico ..... 37
2.8 Diagramas de interacción ..... 38
Conclusiones parciales ..... 40
**CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS.** ..... 41
3.1 Consideraciones sobre la implementación ..... 41
3.2 Modelo de implementación ..... 43
*3.2.1 Diagrama de despliegue* ..... 44
*3.2.2 Diagrama de componentes* ..... 44
*3.2.3 Descripción de los componentes más relevantes* ..... 45
3.3 Estándar de programación utilizado. ..... 46
3.4 Pruebas de Software ..... 48
*3.4.1 Pruebas unitarias* ..... 48
*3.4.2 Pruebas funcionales* ..... 55
*3.4.3 Pruebas de integración* ..... 61
*3.4.4 Pruebas de aceptación* ..... 62
Conclusiones parciales ..... 63
Conclusiones ..... 64
Recomendaciones ..... 65
Referencias Bibliográficas ..... 66
Bibliografía ..... 68

## Índice de Figuras

Figura 1: Etapas del Perfilado de Datos. ..... 6

Figura 2: Mapa conceptual de Pentaho Data Integration. ..... 12

Figura 3: Ejemplo de Trabajo en el PDI. ..... 14

Figura 4: Ciclo de vida de la metodología OpenUP. ..... 19

Figura 5: Modelo de dominio. ..... 25

Figura 6: Diagrama de casos de uso del sistema. ..... 30

Figura 7: Diagrama de clase del diseño del caso de uso realizar análisis general en un flujo de trabajo. ..... 34

Figura 8: Estilo arquitectónico plugin ..... 38

Figura 9: Diagrama de secuencia del caso de uso Realizar análisis general en un flujo de trabajo. ..... 39

Figura 10: Estructura del plugin. ..... 41

Figura 11: Código XML para la incorporación del plugin. ..... 43

Figura 12: Diagrama de componentes del plugin. ..... 44

Figura 13: Grafo de flujo del método Cálculo de varianza. ..... 49

Figura 14: Grafo de flujo del método Añadir valor (String) ..... 50

Figura 15: Grafo de flujo del método Añadir valor (Number). ..... 51

Figura 16: Grafo de flujo del método Añadir valor (Date/Time) ..... 52

Figura 17: Grafo de flujo del método Cálculo de media. ..... 52

Figura 18: Grafo de flujo del método Duplicados ..... 53

Figura 19: Grafo de flujo del método Fecha reciente. ..... 54

Figura 20: Grafo de flujo del método Formato de fecha. ..... 54

Figura 21: Componente para perfilado de datos en funcionamiento a partir de una transformación simple. ..... 61

Figura 22: Datos de análisis arrojados a partir de la ejecución del componente para perfilado de datos. ..... 62

# Índice de tablas

Tabla 1: Comparación entre metodologías ágiles y tradicionales. ..... 16

Tabla 2: Comparación entre algunas de las metodologías ágiles. ..... 17

Tabla 3: Descripción de los actores del sistema. ..... 29

Tabla 4: Descripción del caso de uso Realizar análisis general en un flujo de datos. ..... 31

Tabla 5: Análisis de complejidad ciclomática. ..... 49

Tabla 6: Descripción de las variables correspondientes al caso de prueba para el caso de uso “Realizar la configuración del componente para Pentaho Data Integration”. ..... 56

Tabla 7: Caso de prueba aplicado al caso de uso “Realizar la configuración del componente para Pentaho Data Integration”. ..... 56

Tabla 8: Resumen de los resultados de las pruebas aplicadas. ..... 60

# Introducción

La Universidad de las Ciencias Informáticas (UCI) es un centro de altos estudios surgido a raíz de la necesidad de informatizar el país y desarrollar la industria del software, para contribuir al desarrollo de la sociedad y a la obtención de una nueva fuente de ingresos para la nación. La misma se encuentra compuesta por varios centros de desarrollo, encargados del proceso productivo y del desarrollo de soluciones para diferentes ramas de la sociedad.

El centro de Tecnologías de Gestión de Datos (DATEC), perteneciente a la Facultad 6, cuenta con un departamento dedicado al desarrollo de almacenes de datos, cuyo objetivo o propósito consiste en proporcionar la información de una empresa de forma resumida y con el nivel de detalle adecuado para posteriores análisis que resultan de gran utilidad en el proceso de toma de decisiones gerenciales. Una de las principales tareas que conlleva su elaboración es el perfilado de datos, ya que contribuye a una mejor caracterización de los datos y a un correcto tratamiento de los mismos. Este proceso puede llevarse a cabo antes, durante y después de la integración de los datos, para conocer el estado en que se encuentran y comprobar que presenten la calidad requerida.

Para realizar el perfilado de datos se utilizan un conjunto de herramientas especializadas y opcionalmente el Pentaho Data Integration (PDI). Sin embargo, debido a la naturaleza, formato y ubicación de las fuentes de datos, puede resultar necesario realizar un preprocesado de los datos inicialmente, por lo que generalmente aparecen dos alternativas:

- Realizar el preprocesado con PDI y posteriormente el proceso de perfilado de los datos utilizando las herramientas existentes para dicha actividad.
- Realizar ambas actividades utilizando el PDI.

Ambas alternativas presentan dificultades. En el primer caso, el proceso se encuentra dividido en dos etapas, resultando útil solamente para el perfilado inicial. Esta situación se debe a que las herramientas utilizadas no poseen un alto grado de integración, redundando en la ejecución manual de cada una de las etapas. A lo antes mencionado, se añaden los problemas de compatibilidad que pueden surgir debido a las diferencias en cuanto a arquitectura y manejo de la información entre las herramientas involucradas en dicho proceso.

En el segundo caso, resulta necesario incorporar a los procesos de Integración de datos un gran número de componentes de propósito general, para obtener los resultados esperados. Esta práctica,

añade a todo el proceso un alto grado de complejidad, debido a que el PDI no posee componentes dedicados al perfilado de los datos.

En general, ambas alternativas extienden el proceso de desarrollo de soluciones y aumentan la probabilidad de errores humanos.

Por lo expuesto anteriormente el **problema de la investigación** queda definido en la siguiente interrogante ¿Cómo realizar una caracterización detallada de los datos a través de la herramienta Pentaho Data Integration?

El **objeto de estudio** en la siguiente investigación se plantea como: el perfilado de datos y su **campo de acción**: componente para el perfilado de datos en la herramienta Pentaho Data Integration.

Se define como **objetivo general**: desarrollar un componente para la herramienta PDI, que permita realizar el perfilado de datos.

Para darle cumplimiento al objetivo trazado se determinaron las siguientes preguntas científicas:

¿Cuáles son los fundamentos teóricos del perfilado de datos?

¿Cuáles son las características que debe cumplir el componente del PDI para el proceso de perfilado de datos?

¿Cómo estructurar el proceso de desarrollo del componente del PDI para el proceso de perfilado de datos?

¿El componente del PDI realiza el perfilado de datos?

Se determinaron como **objetivos específicos**:

1) Realizar un análisis sobre las diferentes técnicas y herramientas de perfilado de datos.
2) Fundamentar la selección de las metodologías, herramientas y tecnologías a utilizar en el desarrollo de la solución.
3) Realizar el análisis y diseño del componente de perfilado de datos para Pentaho Data Integration.
4) Implementar el componente para el perfilado de datos.
5) Realizar pruebas al componente para perfilado de datos.

Para dar cumplimiento a los objetivos planteados se trazaron las siguientes **tareas de la investigación**:

1) Elaborar un informe de la arquitectura y funcionamiento de la herramienta Pentaho Data Integration, para el desarrollo del componente.
2) Elaborar un informe con los diferentes tipos de perfilado de datos, para el desarrollo del componente.
3) Caracterizar las metodologías, herramientas y tecnologías a utilizar para el desarrollo del componente.
4) Modelar los conceptos, entidades y eventos más importantes del dominio, para comprender con mayor claridad el negocio existente.
5) Realizar el análisis y diseño del componente para una mayor comprensión de la posible solución.
6) Implementar el componente para representar la caracterización de los datos.
7) Realizar pruebas unitarias, de integración, funcionales y de aceptación para verificar el correcto funcionamiento del componente.

A continuación se realiza una breve descripción de los métodos de investigación aplicados.

**Métodos Lógicos.**

Soporte-Analítico:

A partir de este método se puede estudiar el contenido principal de las técnicas de perfilado de datos y el funcionamiento del PDI para luego descomponer cada una de ellas, obteniendo las más adecuadas y aplicándolas al caso de estudio.

**Métodos Empíricos.**

Investigación – Acción:

Se hace una valoración previa del problema planteado y posteriormente se le da respuesta según los argumentos obtenidos. De esta forma se plantea el problema de la necesidad de realizar el perfilado de datos en la herramienta PDI mediante la utilización de un componente.

**Estructura del trabajo de diploma**

El siguiente trabajo de diploma se ha estructurado en un resumen, introducción, tres capítulos, conclusiones generales, recomendaciones, referencias bibliográficas, bibliografía y anexos.

En el primer capítulo se realiza un análisis del estado del arte de la herramienta Pentaho Data Integration, se seleccionan igualmente las metodologías, herramientas y tecnologías a utilizar en la confección del

componente. En el segundo capítulo se hace alusión a los conceptos relacionados con el negocio, los cuales se reflejan en el modelo de dominio, además, se presenta a través de un diagrama los diversos casos de uso del sistema, así como la especificación de estos últimos. Se selecciona igualmente el patrón arquitectónico que servirá de guía para la implementación del plugin del PDI. En el tercer capítulo se describen los aspectos asociados al desarrollo del plugin como: el diagrama de componentes, la descripción de los componentes más relevantes y los estándares de programación utilizados con el fin de facilitar la implementación. También se detallan las pruebas realizadas para verificar la correspondencia según lo acordado con los especialistas del Departamento Almacenes de Datos.

# Capítulo1: Fundamentos Teóricos.

En el presente capítulo se realizará un estudio sobre el proceso de perfilado de datos y la herramienta Pentaho Data Integration, se describirán además los principales conceptos asociados al dominio del problema. También se realizará un análisis de las tecnologías, metodologías, lenguajes y herramientas a utilizar en el desarrollo del componente.

## 1.1 Conceptos básicos asociados al problema.

Para una mejor comprensión de los términos planteados a lo largo de este capítulo es necesario abordar algunas de las definiciones que se identificaron en la investigación, las cuales se presentan a continuación.

- Componente

Un componente es aquello que forma parte de la composición de un todo. Se trata de elementos que, a través de algún tipo de asociación o contigüidad, dan lugar a un conjunto uniforme.

En la industria del software un componente de software es un elemento de un sistema software que ofrece un conjunto de servicios, o funcionalidades, a través de interfaces definidas. Es una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. (Ariza Rojas, y otros, 2004)

- Perfilado de datos

Por perfilado de datos se entiende el análisis de los datos de los sistemas para entender su contenido, estructura, calidad y dependencias, es decir, un análisis exacto de la estructura y el modelo de datos a tratar. (Soto, 2008)

- Plugin

Un plugin (o plug-in) es un programa de computador que interactúa con otro programa para aportarle una función o utilidad específica. Este programa adicional es ejecutado por la aplicación principal. Los plugins típicos tienen la función de reproducir determinados formatos de gráficos, datos multimedia, codificar o decodificar e-mails, filtrar imágenes de programas gráficos, entre otros.

En la actualidad su uso es muy común. Se utilizan como una forma de expansión para programas de forma modular, de manera que se puedan añadir nuevas funcionalidades sin afectar a las ya existentes ni complicar el desarrollo del programa. (Escobar, y otros, 2006)

## 1.2 Perfilado de datos

El perfilado de datos es sumamente importante en el proceso de integración de datos. Este paso es vital ya que a la hora de plantear un análisis de los datos de origen se puede encontrar en muchas ocasiones que realmente no se tiene ninguna idea acerca de los mismos, ni donde pueden residir algunos de sus problemas.

Para una mejor comprensión, en la figura 3 se muestra las etapas por las que transitan los datos en este proceso.

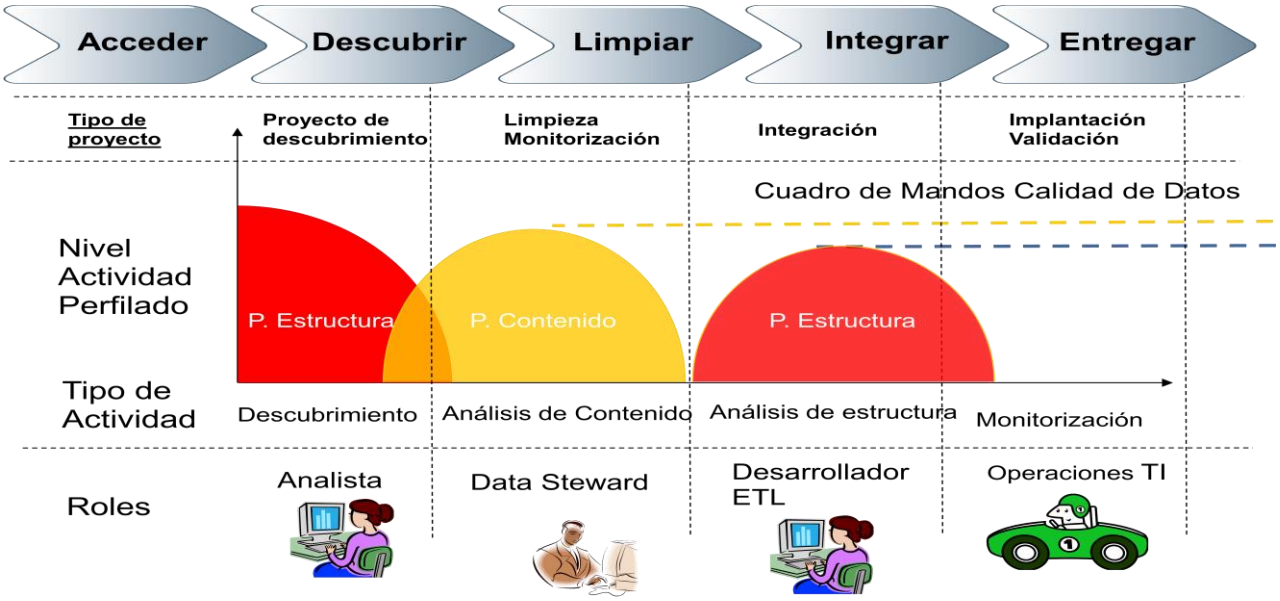



**Figura 1: Etapas del Perfilado de Datos.**

El perfilado de datos se encuentra constituido por dos tipos:

Perfilado de estructura

- El perfilado de estructura consiste en el análisis de los datos sin tener en cuenta su significado
- El análisis se realiza de forma semi-automática y masiva
- Tipos de análisis del Perfilado de Estructura:
  - Perfilado de Columnas

- Perfilado de Dependencias
- Perfilado de Redundancias

Perfilado de contenido

- El perfilado de contenido analiza con profundidad el dato y su significado.
- Requiere una configuración para cada campo a analizar.
- Se combina con el uso de diccionarios, componentes específicos de tratamiento de datos, separadores, etc.

Esta fase proporciona un conocimiento minucioso de la información de origen que debe servir para establecer pautas sobre su posterior normalización, depuración, mejora y tratamiento. Se trabaja a todos los niveles (columna, tabla, modelo). Se identifica qué fuentes pueden dar problemas por cuestiones estructurales para su descarte o modificación con el objetivo de evitar problemas posteriores. En el perfilado de datos se realiza un análisis de las diferentes fuentes de datos origen, esto conlleva a la obtención de los siguientes elementos:

- La estructura de las fuentes origen: tipo de fuente (fichero host, Excel, tabla BBDD, XML), tipo de estructura (bloque fijo, bloque variable), formatos de los campo (tipo de datos, longitud) y nivel de granularidad.
- Contenido de los datos: tipo de información que contiene cada campo de la fuente. Identificar qué información es realmente relevante para el análisis.
- Identificación de claves únicas. Identificación de posibles claves foráneas.
- Validaciones técnicas: integridad, duplicidades, valores obligatorios, nulos.
- Validaciones en base a reglas de negocio. Por ejemplo, rangos de valores límites, valores no posibles, umbrales (edad < 0, código postal de más de n dígitos, número de cuenta corriente de menos de n dígitos).
- Identificación de duplicados, no sólo por validación técnica de clave idéntica, sino de posibles duplicados entre registros que contienen campos con diferentes valores, pero que por error de grabación pueden ser el mismo.
- Estadísticas y distribuciones de datos. Ejemplo al centrarse en una columna efectuar conteos, agregaciones, valores máximos, mínimos, medio, desviaciones, frecuencias, distribución de los valores, rangos de valores, número de valores distintos, ratio de nulos, distribución de longitudes.

- Estadística predictiva: en base a las estadísticas realizadas, obtener el valor más probable, más frecuente, construir tablas de frecuencia en base a las existentes.
- Análisis de los campos descriptivos (descripciones incompletas, errores ortográficos y confusiones fonéticas, caracteres extraños, descripciones en diferentes idiomas), cualquier tipo de error que deba ser corregido o normalizado antes de ser tratada esa información. Reconocimiento de patrones.
- Estudiar la codificación de los campos. Existencia de diferentes codificaciones del mismo campo (ejemplo código cliente o producto, diferente en sistemas de marketing y en sistemas de facturación o en función del ERP que provee la información) y reglas de relación (tablas de normalización) entre ellas.
- Relaciones entre las diferentes fuentes origen. Relaciones entre los campos de una misma fuente. Tipo de relaciones, tipo de dependencias. Identificar restricciones en las relaciones que pueden dar problemas. Detectar relaciones ocultas o no evidentes entre los datos.
- Identificar campos comunes y estandarizados, que deben seguir un patrón establecido, para tratamiento y normalización estándar. Ejemplo: nacionalidad, nombres, direcciones postales, teléfono, e-mail, código de divisa, código de cuenta corriente, código de entidad financiera.
- Determinar el posible impacto de datos que posean una baja calidad. Profundizar en registros específicos con mala calidad de datos para determinar su impacto en el negocio y hallar una solución. Una mala calidad de datos, no impacta los mismos según qué tipo de información provea y puede haber casos de gran impacto en el análisis final de la información.
- Definir las acciones a tomar en los errores detectados. Ejemplo: si se debe establecer código de valor indeterminado para los nulos, qué hacer en los errores de integridad referencial.(Vidal Gil, 2011)

A partir del estudio realizado sobre los tipos de perfilado de datos, se adoptó el perfilado de estructura y dentro de sus tipos de análisis el perfilado de columnas, debido a que es un requisito previo para todos los demás análisis que se deseen llevar a cabo posteriormente y permite la obtención de un resumen de los resultados de cada columna, como las estadísticas e inferencias sobre las características de los datos.

## 1.3 Herramientas existentes para perfilado de datos.

Las actuales herramientas de perfilado de datos proveen una interfaz que facilita la fácil visualización de las relaciones entre los datos y automatizan considerablemente la creación, ejecución y reutilización de reglas de validación y limpieza. Estas herramientas, como es lógico, deben disponer de conectores para acceder a todo tipo de fuentes de datos y todo tipo de plataformas, considerando diferentes infraestructuras tanto locales, remotas como nubes (clouds). Así mismo, deben minimizar el movimiento de datos. (Mínguez Porras, 2011)

Existen varias herramientas para el perfilado de datos, de las cuales la mayoría son propietarias, por lo que su utilización no esta autorizada y su adquisición conlleva un alto concepto de pago. Ante esta situación se decide optar por las herramientas libres, sin embargo aunque se pueden encontrar algunas que realizan dicha actividad, las mismas no cumplen con las necesidades de los especialistas del departamento DATEC.

A continuación se explicarán algunas de las herramientas que se utilizan para el perfilado de datos a nivel mundial, dentro de las cuales se encuentra el Kettle que es de las más utilizadas en el departamento de Almacenes de Datos.

**Informatica Data Explorer:** transforma el concepto que la empresa tiene de los datos. Gracias a ella, los analistas de negocio, los administradores de datos y los desarrolladores de Tecnologías de la Información pueden trabajar conjuntamente para perfilar los datos de todas las aplicaciones y proyectos.
Con Informatica Data Explorer, los administradores de datos y los analistas de negocio podrán perfilar los datos fácilmente y supervisar de forma continua cualquier problema que surja, mediante herramientas basadas en navegador las cuales han sido diseñadas específicamente para ellos. Los desarrolladores pueden detectar y analizar automáticamente los datos empleando reglas predefinidas y un solo entorno de desarrollo unificado para volver a utilizar los resultados del perfilado de datos en todos los proyectos, impulsando así la productividad y eliminando la posibilidad de cometer errores.
Con la utilización de las herramientas que presenta Informatica Data Explorer, se puede lograr:

- Detectar y analizar todas las anomalías de datos en todas las fuentes de datos.
- Descubrir los problemas ocultos de los datos que ponen en riesgo los proyectos.
- Localizar problemas estructurales para evitar los problemas de calidad de datos antes de que crezcan. (The Data Integration Company, 2011)

**Data Profile Task**

Una de las múltiples mejoras que aporta SQL Server 2008 en la parte de ETL con Integration Services es su capacidad para realizar perfilado de datos con su nueva **Data Profile Task**.
La Data Profile Task de *SSIS* funciona seleccionando una tabla de una base de datos *SQLServer 2000* o superior (no funciona con otras bases de datos, ni con otros tipos de fuentes), las opciones de perfilado que se quiera realizar sobre los datos de la tabla, y un fichero XML donde se almacenarán los resultados cuando se ejecute la misma.
Se pueden seleccionar hasta ocho tipos de perfilado, separados en dos categorías.
Perfilados a nivel de columna:

- Distribución de la longitud de los valores
- Porcentaje de valores nulos
- Patrones, expresados mediante expresiones regulares
- Estadísticas de columna: mínimo, máximo, media o desviación standard
- Distribución de los valores, valores diferentes y porcentaje de aparición de cada uno sobre el total de filas

Perfilados a nivel multicolumna:

- Claves candidatas: qué columnas podrían ser clave primaria de la tabla
- Dependencia funcional: los valores de una columna pueden depender de los de otra
- Inclusión de valores: qué columnas podrían ser claves foráneas de otras. (Fernández, 2009)

**Kettle**

Spoon es una Interfaz Gráfica de Usuario (GUI), que permite diseñar transformaciones y trabajos que se pueden ejecutar con las herramientas de Kettle (Pan y Kitchen).

Pan es un motor de transformación de datos que realiza muchas funciones tales como lectura, manipulación, y escritura de datos hacia y desde varias fuentes de datos.

Kitchen es un programa que ejecuta los trabajos diseñados por Spoon en XML (Extensible Markup Language) o en un catálogo de base de datos.Los trabajos normalmente se planifican en modo batch (por lotes) para ejecutarlos automáticamente en intervalos regulares.

Las Transformaciones y Trabajos se pueden describir usando un archivo XML o se pueden colocar en un catálogo de base de datos de Kettle.

Luego Pan o Kitchen pueden leer los datos para ejecutar los pasos que se describen en la Transformación o ejecutar el Trabajo.En resumen, PDI facilita la construcción, actualización, y mantenimiento de Data Warehouses. (Espinosa, 2010)

**Data Cleaner**

DataCleaner es un todo en uno de análisis y optimización de base de datos que además de ser de código libre y estar basado en Java, implementa una variedad de herramientas del análisis de bases de datos y conceptos de limpieza de estas para ayudar a identificar entradas que están obsoletas o que, mayoritariamente, son redundantes. DataCleaner sirve básicamente para eliminar todos los datos superfluos existentes en una base de datos dada. Lo más interesante es que la aplicación es compatible con una amplia gama de plataformas de bases de datos y archivos de datos.

Actualmente, DataCleaner se encuentra integrada a la herramienta Pentaho Data Integration, situación que no resuelve totalmente las necesidades presentadas por los especialistas del centro. Lo anterior se debe a que su utilización en transformaciones no es válida, constituyendo un freno para la continuidad del trabajo con el flujo de datos a tratar, ya que a partir de la misma no se pueden generar más pasos que pudieran ser necesarios para cumplir con los requerimientos definidos con anterioridad por los especialistas. La integración de la herramienta DataCleaner y el PDI solo es válida a partir del desarrollo de trabajos (*jobs*) y no de simples transformaciones.

## 1.4 Pentaho Data Integration

En este epígrafe se llevará a cabo un análisis sobre la herramienta Pentaho Data Integration a partir de su importancia en el entendimiento y solución de la problemática planteada.

En el año 2001, el belga Matt Casters comenzó el desarrollo de una herramienta para uso personal, consciente de las dificultades que había tenido durante su experiencia laboral como constructor de Data Warehouse para la integración de sistemas. Durante los siguientes años, fue desarrollando la herramienta que fue añadiendo nuevas funcionalidades como: acceso a bases de datos, tratamiento de ficheros y componentes hasta llegar a la versión 1.2. El proyecto creció con rapidez y la comunidad se involucró en su desarrollo con mucha actividad, hasta entrar dentro de la órbita de Pentaho (al ser vendido por el autor), que lo incluyó como herramienta ETL (*Extract, Transform, Load*) Extracción, Transformación y

Carga en su suite de productos. Matt Caster ha estado desde entonces trabajando en Pentaho y desarrollando su arquitectura como parte del equipo de Pentaho, interviniendo en las diferentes versiones hasta llegar a la 3.2 y el desarrollo de la nueva versión 4.0, del 2010. Actualmente se encuentra en la versión 5.0. (Espinosa, 2010)

PDI abre, limpia e integra esta valiosa información y la pone en manos del usuario. Provee una consistencia, una sola versión de todos los recursos de información, que es uno de los más grandes desafíos para las organizaciones hoy en día. PDI permite una poderosa ETL (*Extract, Transform, Load*) Extracción, Transformación y Carga.

El uso de la herramienta Kettle permite evitar grandes cargas de trabajo manual frecuentemente difícil de mantener y de desplegar. El nombre de Kettle viene de KDE *Extraction, Transportation, Transformation and Loading Environment*, pues inicialmente la herramienta iba a ser escrita para KDE, el famoso escritorio de GNU/Linux. El producto fue renombrado como Pentaho Data Integration y a partir de ahora será denotado como PDI. (Pentaho Solutions, 2013)

Para la realización del plugin de perfilado de datos y el logro de una mejor comprensión de la herramienta involucrada en el desarrollo, es necesario conocer los elementos que la componen y las relaciones que se establecen entre los mismos, en la figura 1 queda evidenciado su mapa conceptual.

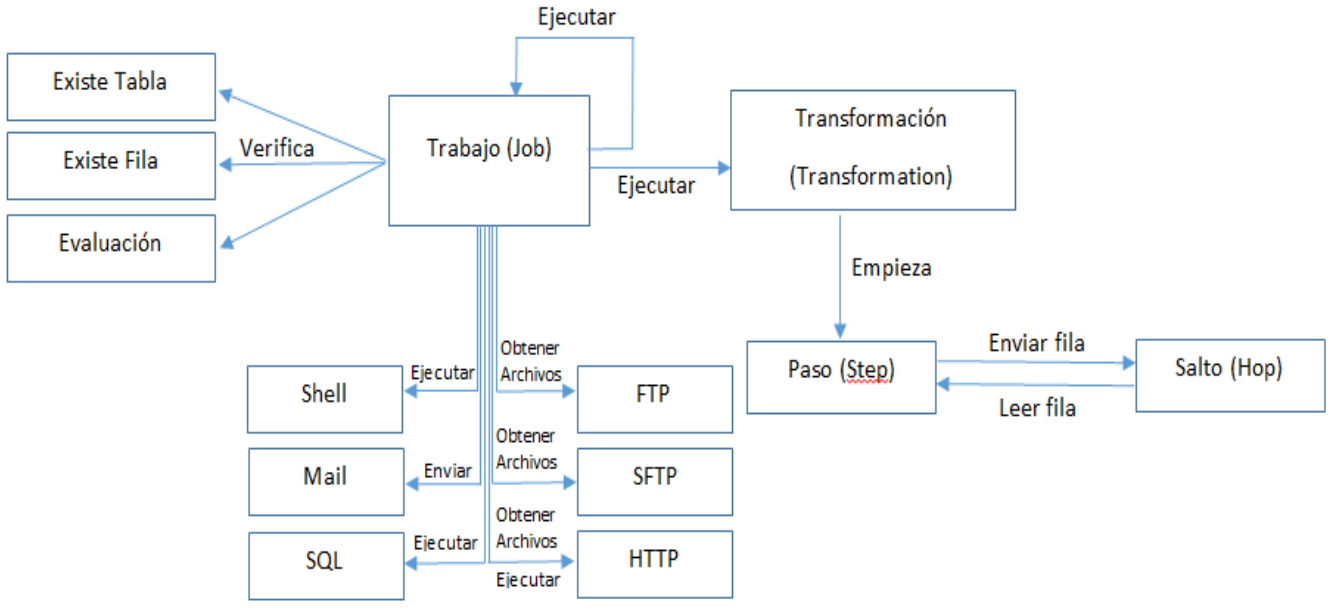



**Figura 2: Mapa conceptual de Pentaho Data Integration.**

**Propiedades básicas**

A parte de ser código abierto (*open source*) y sin costes de licencia, las características básicas de esta herramienta son:

- Entorno gráfico de desarrollo.
- Uso de tecnologías estándar: Java, XML, JavaScript.
- Fácil de instalar y configurar.
- Multiplataforma: Windows, Macintosh, GNU/Linux.
- Basado en dos tipos de objetos: Transformaciones (colección de pasos en un proceso ETL) y trabajos (colección de transformaciones).

Además, PDI incluye un total de cuatro herramientas donde cada una de ellas posee un objetivo específico dentro de los procesos de integración de datos.

Spoon: es la herramienta gráfica que permite el diseño de las transformaciones y trabajos. Incluye opciones para previsualizar y testear los elementos desarrollados. Es la principal herramienta de trabajo del PDI y con la que se construirán y se podrán validar procesos ETL.

Pan: es la herramienta que permite la ejecución de las transformaciones diseñadas en Spoon (bien desde un fichero o desde el repositorio). Posibilita, desde la línea de comandos, preparar la ejecución mediante scripts.

Kitchen: similar a Pan, pero para ejecutar los trabajos o *jobs*.

Carte: es un pequeño servidor web que permite la ejecución remota de transformaciones y *jobs*.

(Diaz Ordaz, 2013)

Los principales elementos que presenta el PDI en el proceso de extracción, carga y transformación de los datos son las transformaciones y los trabajos. A continuación se presenta una breve explicación de los mismos para un mejor entendimiento del funcionamiento de la herramienta.

**Transformaciones**

La transformación es el elemento básico del diseño de los procesos de ETL en el Pentaho Data Integration. Una transformación se compone de pasos (*steps*), que están enlazados entre sí a través de los saltos (*hops*). Los pasos son el elemento más pequeño dentro de las transformaciones mientras que

los saltos constituyen el elemento a través del cual fluye la información entre los diferentes pasos (siempre es la salida de un paso y la entrada de otro).

Existe una amplia colección disponible de pasos, que permite abordar casi cualquier necesidad en el diseño de los procesos de integración de datos. Los pasos están agrupados por categorías y cada uno de ellos está diseñado para cumplir una función determinada. Cada paso tiene una ventana de configuración específica, donde se determinan los elementos a tratar y su forma de comportamiento.

Una transformación no es un programa ni un ejecutable, simplemente es un conjunto de metadatos en XML que le indican al motor del PDI las acciones a realizar. (Espinosa, 2010)

**Jobs**

Por su parte los trabajos o jobs son similares al concepto de proceso, que no es más que un conjunto sencillo o complejo de tareas con el objetivo de realizar una acción determinada. En los jobs se pueden utilizar pasos específicos (que son diferentes a los disponibles en las transformaciones) como recibir un fichero vía ftp (*File Transfer Protocol*, Protocolo de Transferencia de Archivos), enviar un correo y ejecutar un comando, entre otros. Además se puede ejecutar una o varias transformaciones que se hayan diseñado y organizar una secuencia de ejecución de las mismas. Los trabajos estarían en un nivel superior a las transformaciones. (Espinosa, 2010)

Seguidamente se presenta un ejemplo del empleo de los jobs para la ejecución de componentes en la herramienta PDI.

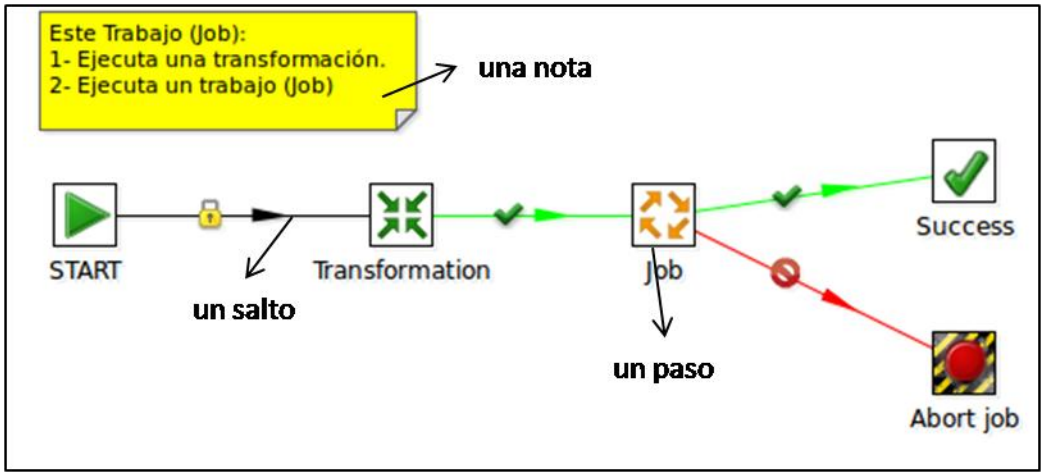



**Figura 3: Ejemplo de Trabajo en el PDI.**

## 1.5 Metodología de desarrollo

Para desarrollar un software se necesita una forma ordenada de trabajo, un proceso que integre y guíe las múltiples facetas del desarrollo y que además, ofrezca criterios para el control y la medición de los productos. A esta clase de procesos se le conoce como metodología de desarrollo. (Hernández, 2010)

Las metodologías se pueden clasificar en ágiles y tradicionales las cuales se detallan a continuación.

**Metodologías Ágiles**

El desarrollo ágil de software emplea métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto-organizado y multidisciplinario. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en lapsos cortos. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, revisión y documentación. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Las principales características de las metodologías ágiles son:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. Las personas que intervienen en el negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.

8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. (Amaro Calderón, y otros, 2007)

**Metodologías Tradicionales**

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, que hacen gran hincapié en la especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas. (Acuña, 2009)

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar. (Acuña, 2009)

Para seleccionar qué clasificación de metodología utilizar se realizó un estudio de las mismas. En la tabla que se presenta a continuación se muestra una comparación entre algunas de las metodologías más utilizadas en la universidad para guiar el proceso de desarrollo del software.

**Tabla 1: Comparación entre metodologías ágiles y tradicionales.**

| Metodología | Tamaño del proceso | Tamaño del equipo | Aprendizaje | Complejidad del problema | Clasificación |
|---|---|---|---|---|---|
| **RUP** | Medio/Extenso | Medio/Extenso | Medio/Lento | Medio/Alto | Tradicional |
| **MSF** | Medio/Extenso | Pequeño /Extenso | Medio/Lento | Medio/Alto | Tradicional |

| | | | | | |
|---|---|---|---|---|---|
| **OpenUP** | Pequeño/Medio | Pequeño | Rápido | Medio/Alto | Ágil |
| **SCRUM** | Pequeño/Medio | Pequeño | Rápido | Medio/Alto | Ágil |
| **XP** | Pequeño/Medio | Pequeño | Rápido | Medio/Alto | Ágil |

Teniendo en cuenta la comparación mostrada anteriormente se define una metodología ágil para guiar el desarrollo del plugin, debido a que es la que más se ajusta a las características del proceso que se quiere realizar y a los recursos disponibles para llevarlo a cabo.

**Tabla 2: Comparación entre algunas de las metodologías ágiles.**

| **Metodología** | **Estabilidad** | **Flexibilidad** | **Rendimiento** | **Diseño** | **Implementación** |
|---|---|---|---|---|---|
| **OpenUP** | Dirigido a la estructura de la arquitectura. | Proporciona lineamientos para todos los componentes que se manipulan en el proyecto. | Perfeccionamiento incesante para obtener retroalimentación y realizar mejoras respectivas. | Establecido en el desarrollo iterativo, ágil e incremental. | Los procesos son adaptables para pequeños grupos de trabajo. |
| **SCRUM** | Aplica la innovación, productividad y competitividad. | Utiliza las sobresalientes técnicas y herramientas para trabajar en equipo. | Equipos agudamente productivos con prioridades definidas. | Orientados a cualquier tipo de situaciones o sistemas de desarrollo de software. Iterativo e incremental. | Proyectos muy complejos. |

| | | | | | |
|---|---|---|---|---|---|
| **XP** | No contiene código duplicado, menor número posible de métodos y clases. | Modelos de implementación y disponibilidad del usuario. | Deja las optimizaciones al final. | Enmiendas puntuales. Funcionalidad mínima. | Proyectos de baja envergadura. |

**Justificación de la metodología propuesta**

OpenUP es una metodología ágil y flexible, conveniente para este tipo de proyecto, ya que se ajusta a grupos de trabajos pequeños, simplifica el desarrollo de software y disminuye en tiempo y costo los esfuerzos invertidos en el proyecto, características acordes a las condiciones que se presentan. Además de cubrir aspectos como la seguridad.

Es una metodología que fomenta el uso de técnicas ágiles y principios, mientras que tiene un ciclo de vida estructurado y probado que hace referencia en la continua entrega de versiones del software, lo que es de agrado para el cliente.

**OpenUP**

OpenUP es un proceso de desarrollo de software mínimamente suficiente, esto quiere decir que incluye solo el contenido fundamental, pues no provee orientación sobre temas en los que el proyecto tiene que lidiar, como son: el tamaño del equipo, el cumplimiento, seguridad, orientación tecnológica entre otras. Sin embargo, OpenUP es completa en el sentido de que manifiesta por completo el proceso de construir un sistema.

Entre las características de Open UP se encuentran:

- El empleo de un modelo de desarrollo incremental.
- Uso de casos de uso y escenarios para un fácil entendimiento de los procesos que se están desarrollando.
- Un eficaz manejo de riesgos.

- Empleo de un diseño basado en la arquitectura.

**Principios de OpenUP**

- Colaborar para sincronizar intereses y compartir conocimiento.
- Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo. Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo. (Olguin Juarez, 2013)

El ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto, en la figura 4 se muestra el que presenta la metodología a utilizar.

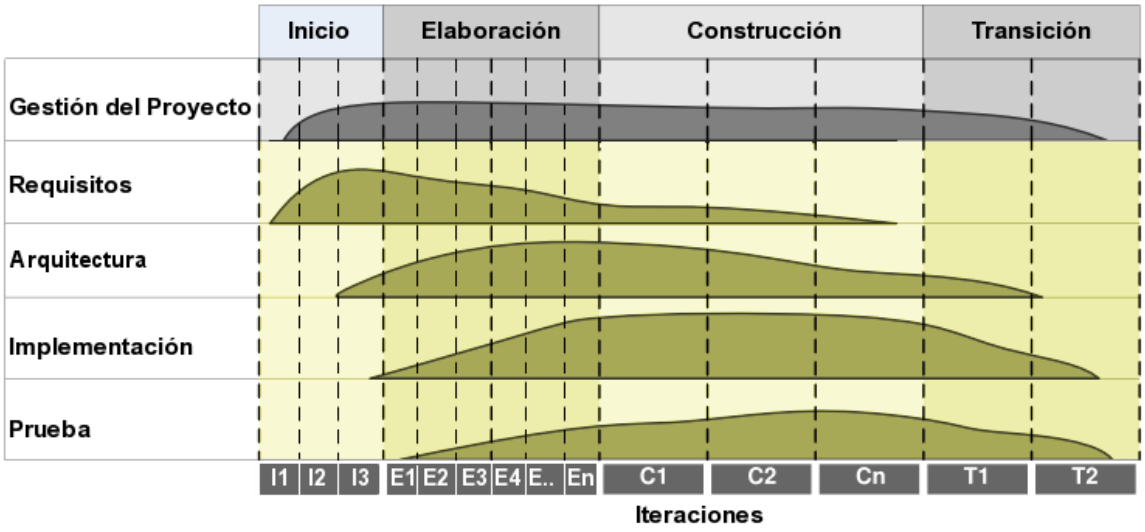



**Figura 4: Ciclo de vida de la metodología OpenUP.**

**Beneficios de OpenUP**

1. Permite disminuir las probabilidades de fracaso.

2. Permite detectar errores tempranos a partir de un ciclo iterativo.

3. Evita la elaboración de documentación y diagramas de iteraciones innecesarios requeridos en las metodologías tradicionales.

4. Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

5. Incrementa las probabilidades de éxito. (Olguin Juarez, 2013)

## 1.6 Herramientas y tecnologías utilizadas

### *1.6.1* ***Herramienta CASE*** (Ingeniería de Software Asistida por Computadora)

Son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida en el desarrollo de sistemas de información, completamente o en alguna de sus fases.
El empleo de herramientas CASE permite integrar el proceso de ciclo de vida:

- Realizan análisis de datos y procesos integrados mediante un repositorio.
- Permiten la generación de interfaces entre el análisis y el diseño.
- Posibilitan la generación del código a partir del diseño.
- Proveen la posibilidad de realizar controles de mantenimiento.

**Visual Paradigm 8.0**

Es una herramienta CASE que emplea el Lenguaje Unificado de Modelado (UML) como lenguaje de modelado. Es fácil de instalar y actualizar, además de que puede ser utilizado durante todo el ciclo de vida del desarrollo de software. Es compatible con los equipos de desarrollo de software en la captura de requisitos, el software de planificación (análisis de casos de uso), ingeniería de código, modelado de clases, el modelado de datos, etc.

Ofrece:

- Entorno de creación de diagramas para UML 2.4.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar y común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Mecanismos para estimar las consecuencias de los cambios y su impacto.
- Disponibilidad en múltiples plataformas. (Visual Paradigm Company, 2013)

Debido a todas las características mencionadas y los beneficios que brinda, especialmente referentes al modelado, se decidió utilizar *Visual Paradigm for UML 8.0* para el modelado de la extensión. Además de

ello, se tuvo en cuenta que esta constituye la principal herramienta que utiliza la Universidad y dentro de ella el centro DATEC para el modelado de software.

### *1.6.2 Lenguaje de programación*

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación. (Mejia, 2012)

**Lenguaje de programación Java**

Java es un lenguaje de plataforma independiente desarrollado por Sun Microsystems en los años 90. El lenguaje toma sintaxis de lenguajes como C y C++, pero a diferencia de este último, combina la sintaxis para programación genérica, estructurada y fue construido desde el principio para ser completamente orientado a objetos y que todo resida en alguna clase. (The Data Integration Company, 2011)

Algunas de las características del lenguaje que se tienen en cuenta a la hora de desarrollar son:

**Interpretado y compilado a la vez:** Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina en la cual se encuentren el intérprete y el sistema de ejecución en tiempo real.

**Indiferente a la arquitectura:** Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar los requisitos de ejecución, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñada para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

**Portable:** la indiferencia a la arquitectura representa sólo una parte de su portabilidad. Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera

que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).

**Dinámico:** El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la red.

**Robusto**: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria. (Alvarez Marañón, 2009)

Debido a que la herramienta PDI se encuentra desarrollada en el lenguaje de programación Java, se adoptó dicho lenguaje para el desarrollo del plugin de perfilado de datos.

### *1.6.3 Entorno de desarrollo integrado*

Un entorno de desarrollo integrado, es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

**IDE Eclipse-Indigo 4.07**

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar Entornos de Desarrollo Integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus. El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando

otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. Su arquitectura de plugin permite desarrollar cualquier extensión deseada en el ambiente. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación. (Márquez, y otros, 2012)

Este IDE fue el utilizado para el desarrollo del código fuente del PDI, por lo que se hace uso del mismo para la construcción del plugin de perfilado de datos.

### *1.6.4 Pentaho Data Integration 4.2.1*

Es una herramienta que permite la realización de los procesos de ETL de forma gráfica, por lo que los mismos quedan plasmados en forma de flujos. La herramienta es muy intuitiva y de fácil uso, conceptualmente muy sencilla y potente.

Ventajas:

- Es una de las herramientas libres más antiguas, poseedora de una gran cantidad de usuarios y una nueva dirección por parte del soporte técnico de Pentaho.
- La licencia pública de Mozilla (organización sin ánimo de lucro que produce software libre) permite vincular Kettle en otros productos sin el pago de otras licencias.

Desventajas:

- No tiene módulo especializado ni acuerdos con ningún proveedor que le proporcione calidad de datos. (Espinosa, 2010)

## Conclusiones parciales

En este capítulo se realizó un estudio acerca de la herramienta PDI con el objetivo de entender su funcionamiento y la forma más factible de solucionar el problema que se plantea. Se analizaron las principales metodologías de desarrollo de software, con el fin de seleccionar la que más se ajusta al sistema teniendo en cuenta las características del mismo. Igualmente fueron definidas las herramientas y tecnologías que serán utilizadas en el proceso de desarrollo del componente, dentro de las cuales cabe destacar como IDE de desarrollo el Eclipse 4.07, como lenguaje de programación Java y para el modelado se usó Visual Paradigm en su versión 8.0. Fueron definidos los métodos de investigación científica y demás elementos básicos en el ciclo de vida del desarrollo del plugin para Pentaho Data Integration.

# CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA.

En el presente capítulo se identifican los requisitos funcionales y no funcionales. Además se muestra el diseño de la solución propuesta así como las definiciones de los elementos que intervienen en la misma y se da cumplimiento a la elaboración de los artefactos que arroja la metodología seleccionada.

## 2.1 Modelo del dominio

Un modelo del dominio es una representación de las clases conceptuales del mundo real. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema, por lo cual este puede ser tomado como el punto de partida para el diseño del sistema.

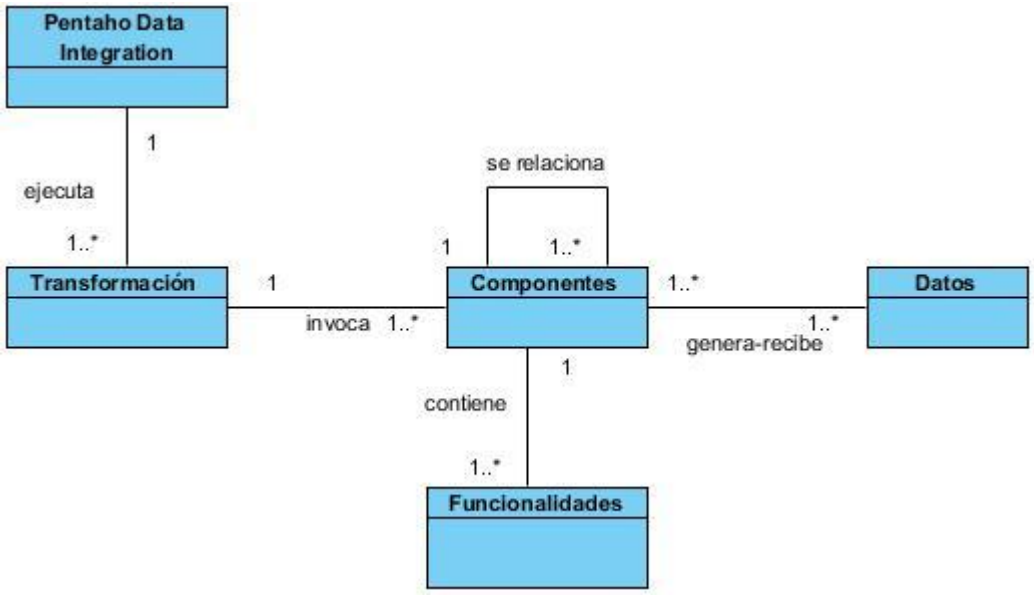



**Figura 5: Modelo de dominio.**

### *2.1.1 Descripción de las clases del dominio*

**Pentaho Data Integration:** representa la herramienta que se encarga de regir el proceso de ETL.

**Transformación:** representa un flujo de trabajo dentro del proceso de ETL.

**Componentes:** representa un conjunto de funcionalidades enfocadas a obtener un resultado específico.

**Datos:** es la información con la que interactúa el componente.

**Funcionalidades:** representan los elementos necesarios para lograr el correcto funcionamiento del componente y lograr el resultado esperado.

## 2.2 Propuesta de solución

Teniendo en cuenta la problemática existente y con la tarea de cumplir los objetivos trazados para dar solución a la misma, se propone realizar un plugin para la herramienta Pentaho Data Integration que, a partir de flujos de datos seleccionados, sea capaz de realizar el perfilado de datos empleando cuatro tipos de análisis.

## 2.3 Especificación de los requisitos

### *2.3.1 Requisitos funcionales*

Los requerimientos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Constituyen el punto de partida para identificar qué debe hacer el sistema y se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. (EVA, 2007-2008)

**RF1** Configurar el componente para Pentaho Data Integration.

El sistema debe permitir configurar el componente para Pentaho Data Integration.

**RF2** Calcular el total de valores dado un campo en el flujo de datos.

El sistema debe permitir calcular el total de valores dado un campo en el flujo de datos.

**RF3** Calcular la cantidad de valores nulos dado un campo en el flujo de datos.

El sistema debe permitir calcular la cantidad de valores nulos dado un campo en el flujo de datos.

**RF4** Calcular la cantidad de valores duplicados dado un campo en el flujo de datos.

El sistema debe permitir calcular la cantidad de valores duplicados dado un campo en el flujo de datos.

**RF5** Calcular la cantidad de valores únicos para los campos de tipo texto (String) en el flujo de datos.

El sistema debe permitir calcular la cantidad de valores únicos para los campos de tipo texto (String) en el flujo de datos.

**RF6** Calcular la cantidad de caracteres del valor de menor longitud para los campos de tipo texto (String) en el flujo de datos.

El sistema debe permitir calcular la cantidad de caracteres del valor de menor longitud para los campos de tipo texto (String) en el flujo de datos.

**RF7** Calcular la cantidad de caracteres del valor de mayor longitud para los campos de tipo texto (String) en el flujo de datos.

El sistema debe permitir calcular la cantidad de caracteres del valor de mayor longitud para los campos de tipo texto (String) en el flujo de datos.

**RF8** Determinar el valor máximo dado un campo numérico en el flujo de datos.

El sistema debe permitir determinar el valor máximo dado un campo numérico en el flujo de datos.

**RF9** Determinar el valor mínimo dado un campo numérico en el flujo de datos.

El sistema debe permitir determinar el valor mínimo para cada campo numérico en el flujo de datos.

**RF10** Calcular la sumatoria de todos los valores dado un campo numérico en el flujo de datos.

El sistema debe permitir calcular la sumatoria de todos los valores dado un campo numérico en el flujo de datos.

**RF11** Calcular el valor promedio dado un campo numérico en el flujo de datos.

El sistema debe permitir calcular el valor promedio dado un campo numérico en el flujo de datos.

**RF12** Calcular el valor de la varianza dado un campo numérico en el flujo de datos.

El sistema debe permitir calcular el valor de la varianza para cada campo numérico en el flujo de datos.

**RF13** Determinar el valor mínimo dado un campo de tipo Fecha/Tiempo (Date/Time) en un flujo de datos.

El sistema debe permitir determinar la fecha más antigua para los campos de tipo Fecha/Tiempo (Date/Time) en un flujo de trabajo.

**RF14** Determinar el valor máximo dado un campo de tipo Fecha/Tiempo (Date/Time) en un flujo de datos.

El sistema debe permitir determinar el formato para los campos de tipo Fecha/Tiempo (Date/Time) en un flujo de datos.

**RF15** Determinar el formato de fecha dado un campo de tipo texto (String) a partir del análisis Fecha/Tiempo (Date/Time) en un flujo de datos.

El sistema debe permitir determinar el formato de fecha dado un campo de tipo texto (String) a partir del análisis Fecha/Tiempo (Date/Time) en un flujo de datos.

### *2.2.2 Requisitos no funcionales*

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. (EVA, 2007-2008)

**Requisitos de Usabilidad.**

**RNF1** El tiempo de entrenamiento requerido para que usuarios normales y avanzados sean productivos operando el sistema será corto.

**RNF2** La interfaz debe contar con mensajes contextuales asociados a los elementos que la componen.

**RNF3** Facilidad de uso por parte de los usuarios: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.

**Requisitos de Eficiencia.**

**RNF4** El tiempo de ejecución de los análisis será proporcional al volumen de datos a analizar.

**RNF5** Requisitos mínimos necesarios para el funcionamiento de la aplicación.

Procesador: Celeron, 2.0 GHz.

1. Memoria RAM: 128 Mb.
2. Espacio libre en disco duro: 200 Mb.

**Requisitos de restricción de diseño.**

**RNF6** Requisitos de software

1. Lenguaje de programación Java.
2. Eclipse-Indigo 4.07 como IDE de desarrollo.
3. Visual Paradigm V8.0 como herramienta CASE.

4. La herramienta PDI.
5. Máquina Virtual de Java 1.5 o superior.

**Requisitos de Soporte**

**RFN7** El plugin una vez integrado a la herramienta Pentaho Data Integration, se podrá utilizar en diferentes sistemas operativos por ser Pentaho Data Integration una herramienta multiplataforma.

## 2.4 Definición de los casos de uso

Según Pressman un caso de uso es*,"… simplemente, un texto escrito que describe el papel de un actor que interactúa."*

Los casos de usos identificados son los siguientes:

1. Realizar la configuración del componente para Pentaho Data Integration.
2. Realizar análisis general en un flujo de datos.
3. Realizar análisis de tipo texto (String).
4. Realizar análisis de tipo numérico (Number).
5. Realizar análisis de tipo fecha/tiempo (Date/Time).

### *2.4.1 Descripción de los actores del sistema*

**Tabla 3: Descripción de los actores del sistema.**

| Actor | Descripción |
|---|---|
| Usuario | Persona que interactúa con el componente. |
| Pentaho Data Integration | Herramienta que ejecuta el componente. |

En la figura 6 se muestra el diagrama de casos de uso del sistema.

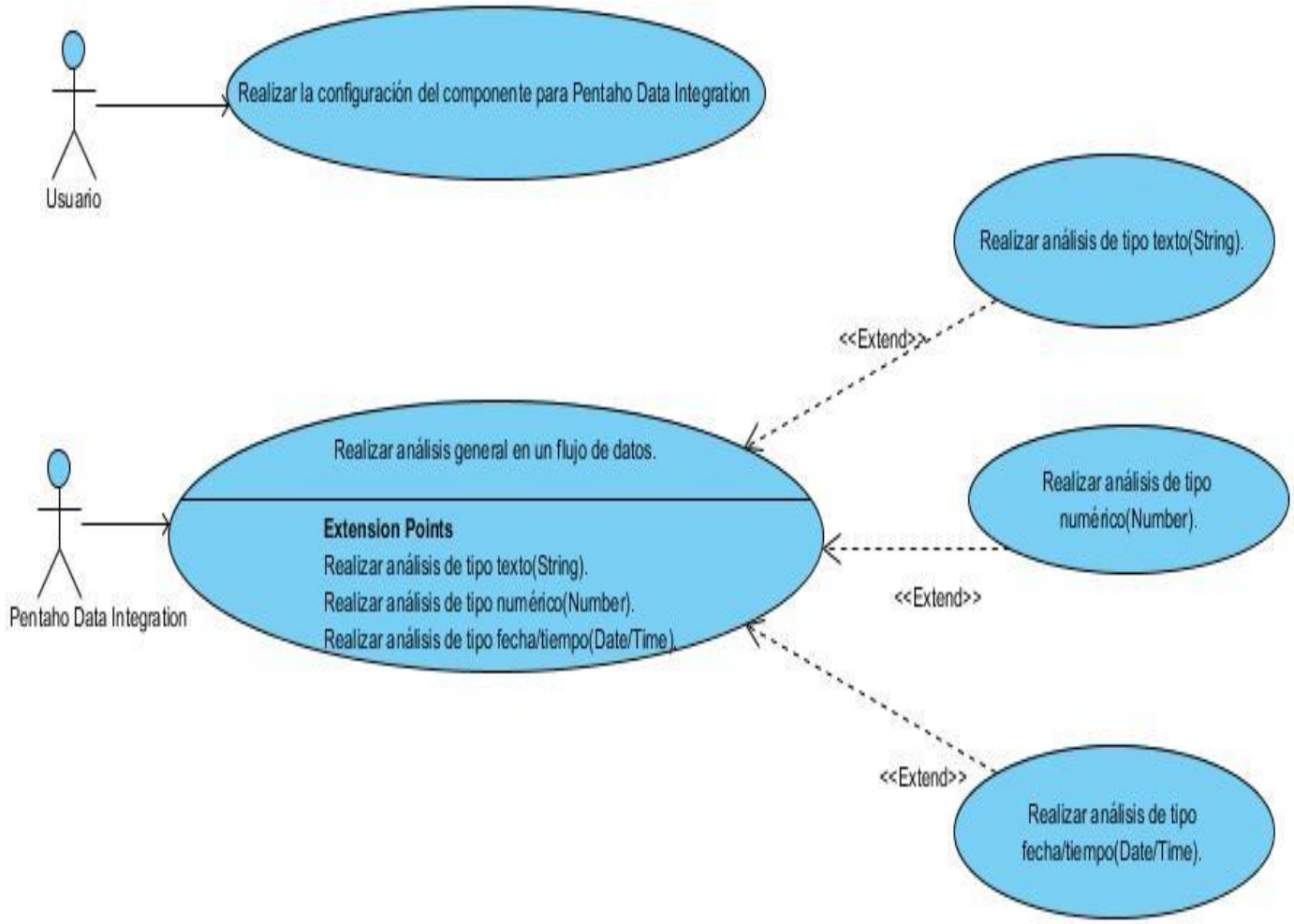



**Figura 6: Diagrama de casos de uso del sistema.**

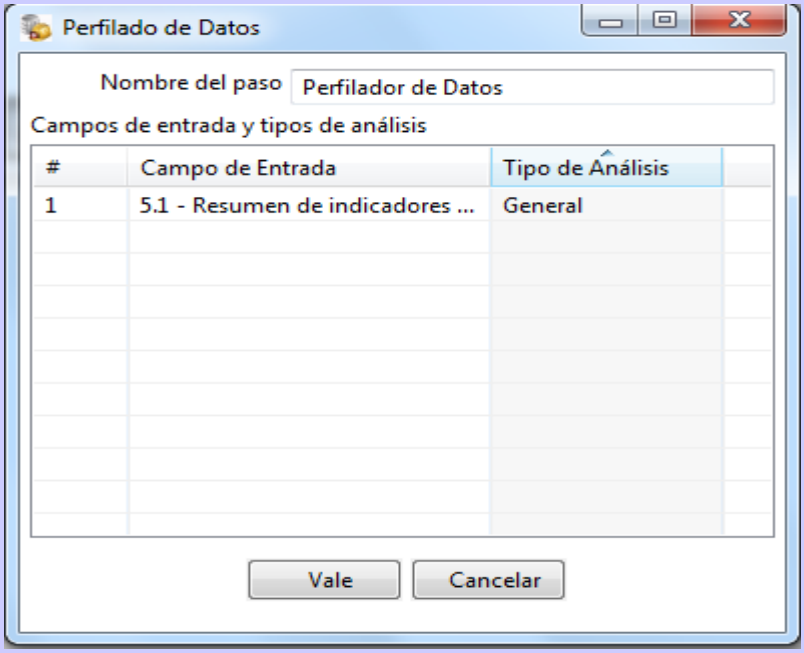
A continuación se realizará una descripción de los casos de uso definidos para un mayor entendimiento del sistema.

### *2.4.2 Descripción del caso de uso Realizar análisis general en un flujo de datos.*

**Tabla 4: Descripción del caso de uso Realizar análisis general en un flujo de datos.**

| | |
|---|---|
| Objetivo | Realizar análisis de tipo general |
| Actores | Pentaho Data Integration. |
| Resumen | El caso de uso inicia cuando se ejecuta la transformación. El sistema captura el flujo de datos e internamente la plataforma los analiza. El caso de uso termina cuando se envían los datos al próximo paso dentro del flujo de datos. |
| Complejidad | Alta |
| Prioridad | Alta |
| Precondiciones | El caso de uso realizar la configuración del componente para Pentaho Data Integration tendría que haber sido llevado a cabo.<br>Tendría que haber sido seleccionada la opción de realizar análisis de tipo general al flujo de trabajo. |
| Poscondiciones | Realizar análisis general. |
| Flujo de eventos | |

| **Flujo básico:** Realizar análisis general al flujo de datos. | | |
|---|---|---|
| | Actor | Sistema |
| 1. | 1-Ejecuta la transformación. | |
| 2. | | 2-Invoca el método init() de la clase DataProfileStep para inicializar la ejecución del paso. |
| 3. | | 3-Se invoca el método run() para detectar errores y activar el log del componente. |
| 4. | | 4-Invoca el método processRow() para procesar los elementos de tipo meta y data resultantes del análisis realizado. |
| 5. | | 5-Realiza un análisis de tipo general de los datos. |
| 6. | | 6-Invoca el método dispose() para finalizar la ejecución del paso y enviar el flujo de datos analizado hacia el siguiente paso de la transformación. |
| 7. | Visualiza los resultados de la ejecución del paso. | |

| | | |
|---|---|---|
| Relaciones | CU Incluidos | Ninguno |
| | CU Extendidos | CU Realizar análisis de tipo texto (String).<br>CU Realizar análisis de tipo numérico (Number).<br>CU Realizar análisis de tipo fecha/tiempo (Date/Time). |
| Requisitos no funcionales | RNF1,RNF4 | |
| Asuntos pendientes | Ninguno | |
| Prototipo de interfaz de usuario |  | |



## 2.5 Diagramas de clases de diseño

Un diagrama de clases de diseño (DCD) representa las especificaciones de las clases e interfaces en una aplicación. A continuación se muestra el diagrama de clases de diseño del caso de uso Realizar análisis general en un flujo de trabajo.

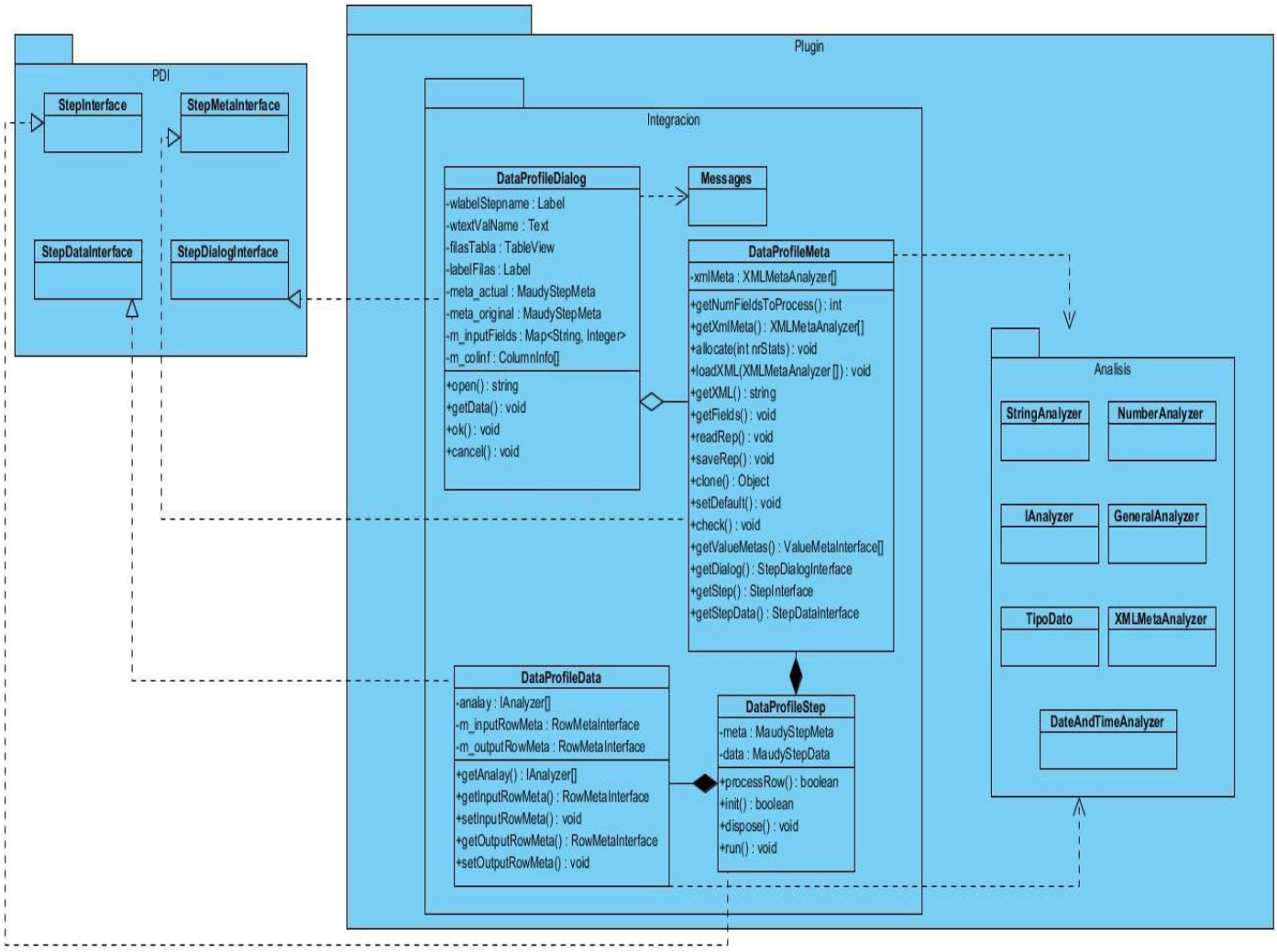



**Figura 7: Diagrama de clase del diseño del caso de uso realizar análisis general en un flujo de trabajo.**

El diagrama de clase del diseño del caso de uso Realizar análisis general en un flujo de datos, mostrado anteriormente, está compuesto por cuatro paquetes, donde en el paquete PDI están contenidas las Interfaces de Programación de Aplicaciones (API) para el desarrollo de plugins en la herramienta. En el paquete Plugin se encuentran las cuatro clases básicas, las cuales heredan de las API del PDI, así como la base de mensajes utilizada por la interfaz de la aplicación y el paquete de funcionalidades para realizar los distintos tipos de análisis y la configuración de los metadatos.

## 2.6 Patrones utilizados

**Patrones GRASP**

En el diseño orientado a objetos, los Patrones Generales de Asignación de Responsabilidad de Software (GRASP) como su nombre lo indica son patrones generales de software para asignación de responsabilidades. Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendables en el diseño de software.

Alta Cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. En el caso del componente en cuestión cada una de las clases (DataProfileStep, DataProfileMeta, DataProfileData, DataProfileDialog) se encarga de realizar tareas específicas que logran que el plugin funcione como un todo.

Creador: se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando.

B contiene a A.

B es una agregación (o composición) de A.

B almacena a A.

B tiene los datos de inicialización de A (datos que requiere su constructor).

B usa a A. (Caso que se presenta en el desarrollo del plugin).

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. En el caso particular del componente a desarrollar se presentan como ejemplos de este patrón las clases DataProfileStep y DataProfileDialog.

Experto: la responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Hay que tener en cuenta que esto es aplicable mientras se estén considerando los mismos aspectos del sistema. Un ejemplo del uso de este patrón es la clase DataProfileDialog.

Controlador: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.

Bajo Acoplamiento: Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia. (Astudillo, y otros)

Se tuvo en cuenta al modelar el sistema los conceptos de bajo acoplamiento para evitar las dependencias excesivas, y la alta cohesión tratando de que cada clase realice labores únicas y bien relacionadas, siempre con la intención de lograr un punto de equilibrio entre ambos. Además, el patrón experto para la realización de las tareas, siendo responsabilidad de la clase que cuenta con los datos involucrados, y el controlador para manejar los eventos del sistema. La clase DataProfileStep es un claro ejemplo de este tipo de patrón evidenciado en el plugin para el perfilado de datos.

**Patrones GoF**

Conocidos en muchas ocasiones por el termino de patrones GoF (Gang of Four), hace referencia a los cuatro autores del libro *"Design Patterns: Elements of Reusable Object-Oriented Software"*: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

El GoF describe 23 patrones, clasificados según tres tipos o propósitos:

Creación: Cómo se puede crear un objeto. Habitualmente esto incluye aislar los detalles de su creación, de forma que el código no dependa de los tipos de objeto que se presentan y por lo tanto, no tenga que cambiarlo cuando añada un nuevo tipo de objeto.

Estructura: Esto afecta a la manera en que los objetos se conectan con otros objetos para asegurar que los cambios del sistema no requieren cambiar esas conexiones. Los patrones estructurales suelen imponer las restricciones del proyecto.

Comportamiento: Objetos que manejan tipos particulares de acciones dentro de un programa. Éstos encapsulan procesos que se quieren ejecutar, permiten interpretar un lenguaje, completar una petición, moverse a través de una secuencia (como en un iterador) o implementar un algoritmo.

En el caso específico del plugin para Pentaho Data Integration se utilizaron los siguientes patrones GoF:

Mediator: A modo de elemento en el medio, define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

1. Se gana un bajo acoplamiento entre los objetos.
2. Cada objeto es responsable solamente de su comportamiento.
3. Un objeto no tiene por qué conocer a los objetos con los que se relaciona, de esto se encarga el mediador.
4. Si se incorpora algún nuevo objeto al sistema, basta con registrarlo en el mediador, no es necesario modificar código en los demás objetos.
5. Los objetos pueden ser reutilizados.

Un ejemplo del empleo de este patrón se puede evidenciar en la clase IAnalyzer.

Iterator: Este patrón se basa en el recorrido de todos los elementos de una colección, una vez cada uno, y sin seguir un orden en concreto. Implementa una interfaz que devuelve en cada momento un apuntador a un elemento de la colección, y que además, posibilita mover este apuntador de principio a fin, permitiendo de esta manera acceder a todos los demás.

1. Es relativamente fácil el acceso a los elementos de la colección, ya que es esta quien se encarga de brindar esta funcionalidad de manera transparente al que accede a los datos.
2. La lista puede sufrir cambios en su estructura interna, pero el acceso a los datos será siempre igual.
3. Se puede recorrer la misma colección varias veces al mismo tiempo utilizando más de un iterador.

Un ejemplo del empleo de este patrón se puede evidenciar en la clase IAnalyzer.

## 2.7 Patrón arquitectónico

Estilo Arquitéctonico Plug-in

Los estilos arquitectónicos definen las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software. (Reynoso, 2005)

Un plug-in (extensión) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e

interactúa por medio de la Interfaz Programada de la Aplicación (por sus siglas en inglés API) (Johannes Mayer, 2006).

Dicho estilo no contiene ninguna variante arquitectónica, pues se ve de forma independiente. Se enfoca en explicar cómo se puede diseñar una aplicación con el fin de soportar varios plug-in, permitiendo que la misma se extienda en tiempo de ejecución mediante la carga dinámica de módulos o clases que no conoce durante la compilación, ofreciendo una escalabilidad de forma rápida y poco costosa.

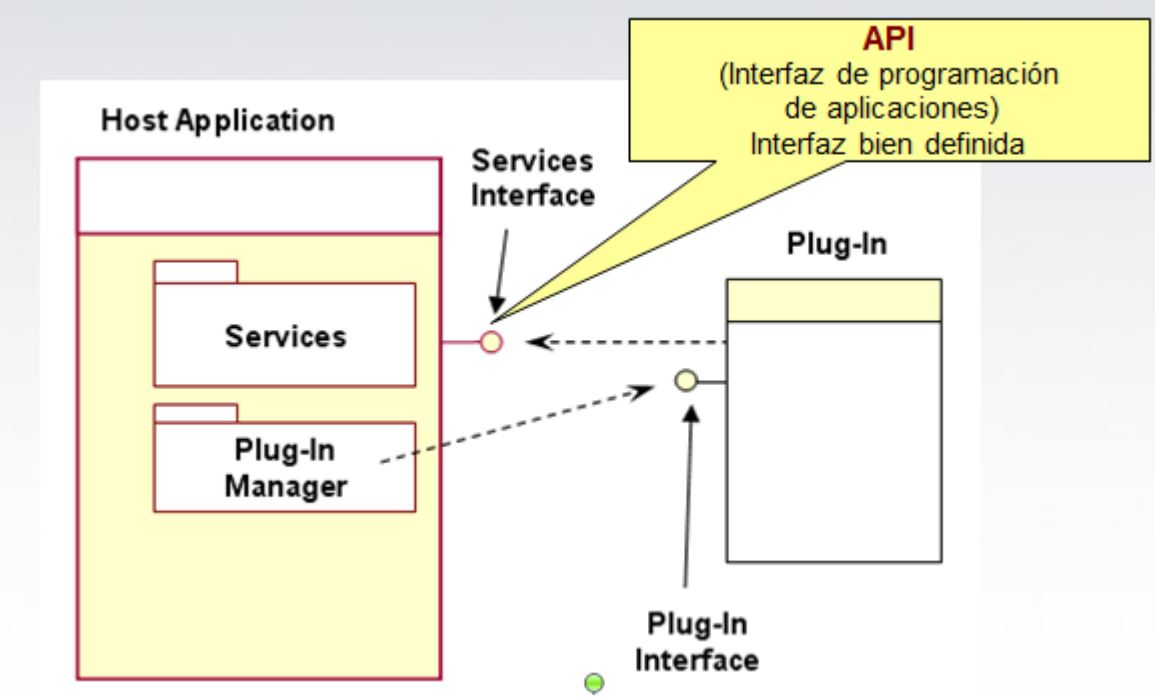



**Figura 8: Estilo arquitectónico plugin.**

## 2.8 Diagramas de interacción

Todos los sistemas tienen una estructura estática y un comportamiento dinámico, y el UML proporciona diagramas para capturar y describir ambos aspectos.

Los diagramas de interacción describen el comportamiento de un sistema, para demostrar cómo los objetos interactúan dinámicamente en diferentes momentos durante la ejecución del sistema.

Los objetos dentro de un sistema se comunican unos con otros, enviándose mensajes. Un mensaje es justo una operación donde un objeto llama a otro objeto. Así pues la dinámica de un sistema se refiere a cómo los objetos dentro del sistema cambian de estado durante el ciclo de vida del mismo y también a

cómo dichos objetos colaboran a través de la comunicación. En el primer caso se utilizan los diagramas de estado y los diagramas de actividad. En el segundo caso, la comunicación entre los objetos se representa mediante los diagramas de interacción, que a su vez agrupan a dos tipos de diagramas: secuencia y colaboración. (Otero Vidal, 2008)

**Diagrama de secuencia**

Los diagramas de secuencia muestran la interacción de un conjunto de objetos a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes. Es decir, proporciona la interacción entre los objetos, que se sucede en el tiempo, para un escenario específico durante la ejecución del sistema. (Otero Vidal, 2008)

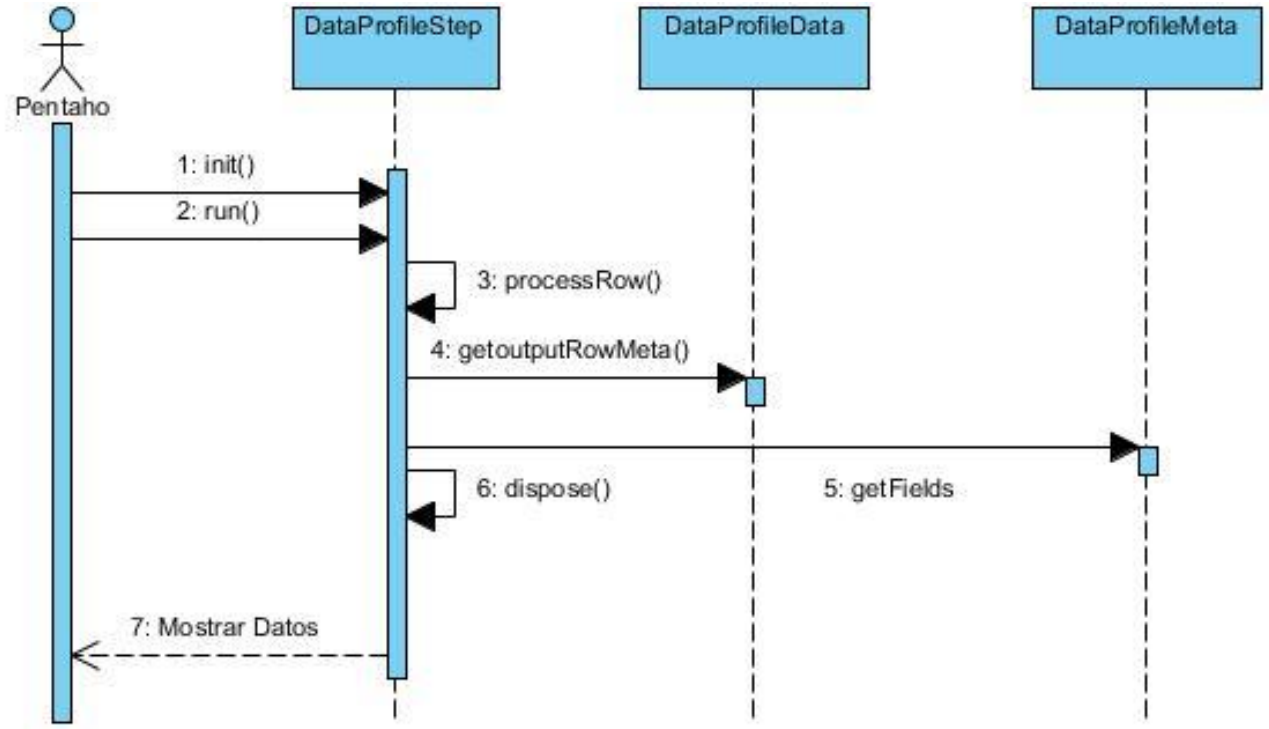



**Figura 9: Diagrama de secuencia del caso de uso Realizar análisis general en un flujo de trabajo.**

## Conclusiones parciales

En este capítulo se efectuó un análisis de los conceptos relacionados con el problema planteado, empleándose para ello, un modelo de dominio que sirvió de apoyo para lograr una mejor familiarización con los elementos involucrados en el desarrollo. Además, se seleccionaron patrones para orientar mejor el desarrollo y fueron descritos los requisitos necesarios para dar vida a las funcionalidades del plugin. De igual manera se plasmaron los artefactos que dan respuesta a la metodología seleccionada y que brindarán un mayor entendimiento al problema y su solución.

# CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS.

En el presente capítulo se define el modelo de implementación que muestra el diseño de la solución, así como el diagrama de componentes del plugin. Se describen los estándares de programación utilizados en la implementación y se especifican los tipos de pruebas realizadas con el objetivo de comprobar las funcionalidades del plugin y verificar que los resultados sean los esperados.

## 3.1 Consideraciones sobre la implementación

Para la implementación del plugin es necesario conocer la estructura de los componentes del PDI, así como la arquitectura del PDI y la configuración del archivo plugin.xml, los cuales se abordarán seguidamente:

**Estructura del plugin**

El proyecto está compuesto por varios paquetes. En la siguiente figura se muestra la estructura del plugin:

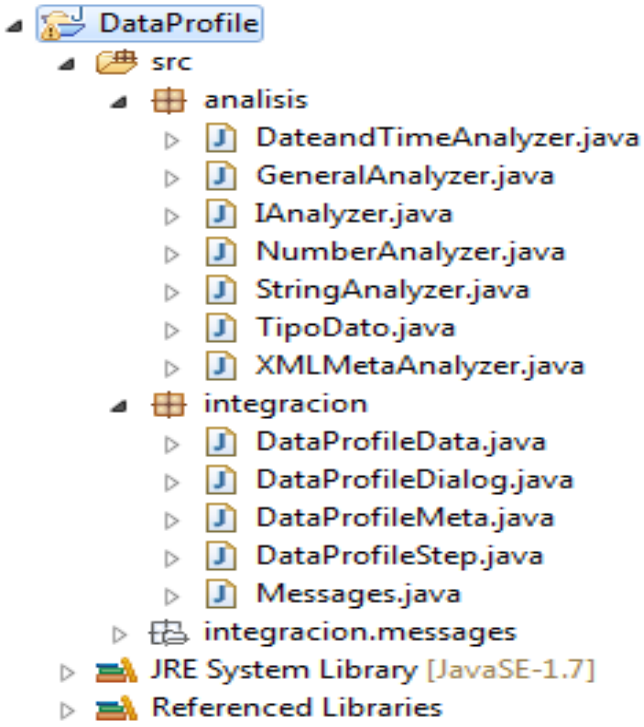



**Figura 10: Estructura del plugin.**

El paquete **Integracion** es el encargado de la integración del flujo de información con la herramienta PDI.

**Arquitectura del plugin para Pentaho Data Integration**

**DataProfileData:** la clase de datos se utiliza para el almacenamiento de datos únicos a un hilo de ejecución, cuando el complemento se ejecuta. Aquí es donde las conexiones de base de datos almacenan en caché los identificadores de archivos, y demás elementos necesarios durante la ejecución. La clase implementa la interface StepDataInterface.

**DataProfileMeta:** la clase implementa la StepMetaInterface. Es responsable de sostener y fabricar en serie las escenas escogidas para un caso particular del paso. Sostiene el nombre del paso y el nombre del campo del rendimiento para el paso básico.

**DataProfileStep:** la clase paso (step) implementa el StepInterface. Las instancias de esta clase hacen el procesamiento de filas real cuando se ejecuta la transformación. Cada hilo de ejecución está representado por una instancia de esta clase. Se administra las instancias de las clases de datos y meta, cuando se ejecuta.

**DataProfileDialog:** la clase de diálogo implementa la interfaz de usuario del paso. Se muestra un cuadro de diálogo que permite al usuario configurar el comportamiento del paso a su agrado. Esta clase de diálogo está estrechamente relacionado con la clase meta, que realiza un seguimiento de los ajustes seleccionados. (Chodnicki, 2010)

Además del código, los plugin tienen otras características que permiten su unión con la herramienta, accesible a través de la interfaz del usuario. Existe un fichero (plugin.xml) que guarda toda la información externa necesaria para la visualización del plugin.

**Escribiendo el archivo plugin.xml**

La configuración del archivo plugin.xml es bastante simple. Su función principal es decirle a la herramienta PDI la clase que inicia el proceso (la clase meta), la descripción y localización del plugin.

```
-<plugin id="DataProfile" iconfile="DataProfile.png" description="Perfilador de Datos" tooltip="Este es un plugin para realizar el proceso de perfilado de datos" category="Transform"
 classname="integracion.DataProfileMeta">
  -<libraries>
     <library name="DataProfile.jar"/>
   </libraries>
  -<localized_category>
     <category locale="en_US">Transform</category>
   </localized_category>
  -<localized_description>
     <description locale="en_US">Perfilador de Datos</description>
   </localized_description>
  -<localized_tooltip>
    -<tooltip locale="en_US">
       Este es un plugin para realizar el proceso de perfilado de datos
     </tooltip>
   </localized_tooltip>
</plugin>
```

**Figura 11: Código XML para la incorporación del plugin.**

El **identificador** para el plugin debe ser globalmente único, y no debe cambiarse, debido a que es utilizado en el proceso de serialización. El archivo del ícono es una imagen de extensión png (Gráfico de Red Portátil). La **descripción** constituye el nombre del paso, siendo la forma en que se representa dentro del árbol de menú en el PDI. La **categoría** es el nombre de la carpeta del árbol en la que aparece el plugin. (Chodnicki, 2010)

## 3.2 Modelo de implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

Un diagrama de implementación muestra:

- Las dependencias entre las partes de código del sistema (diagramas de componentes).

- La estructura del sistema en ejecución (diagrama de despliegue).

### *3.2.1 Diagrama de despliegue*

Un diagrama de despliegue constituye una representación física de las relaciones que existen entre los componentes de hardware y software en el sistema, a través del cual es representada la configuración de los elementos de procesamiento y los componentes de software en tiempo de ejecución. Solo los componentes que son utilizados en tiempo de compilación deben mostrarse en el diagrama de despliegue. Los componentes del plugin son compilados por la herramienta PDI una vez iniciada la aplicación, formando parte de un único nodo físico de procesamiento. Por las razones antes expuestas se determinó realizar el modelo de implementación mediante el diagrama de componentes. (Microsoft, 2014)

### *3.2.2 Diagrama de componentes*

En el diagrama de componentes se muestran los elementos de diseño de un sistema de software. El mismo permite visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces.

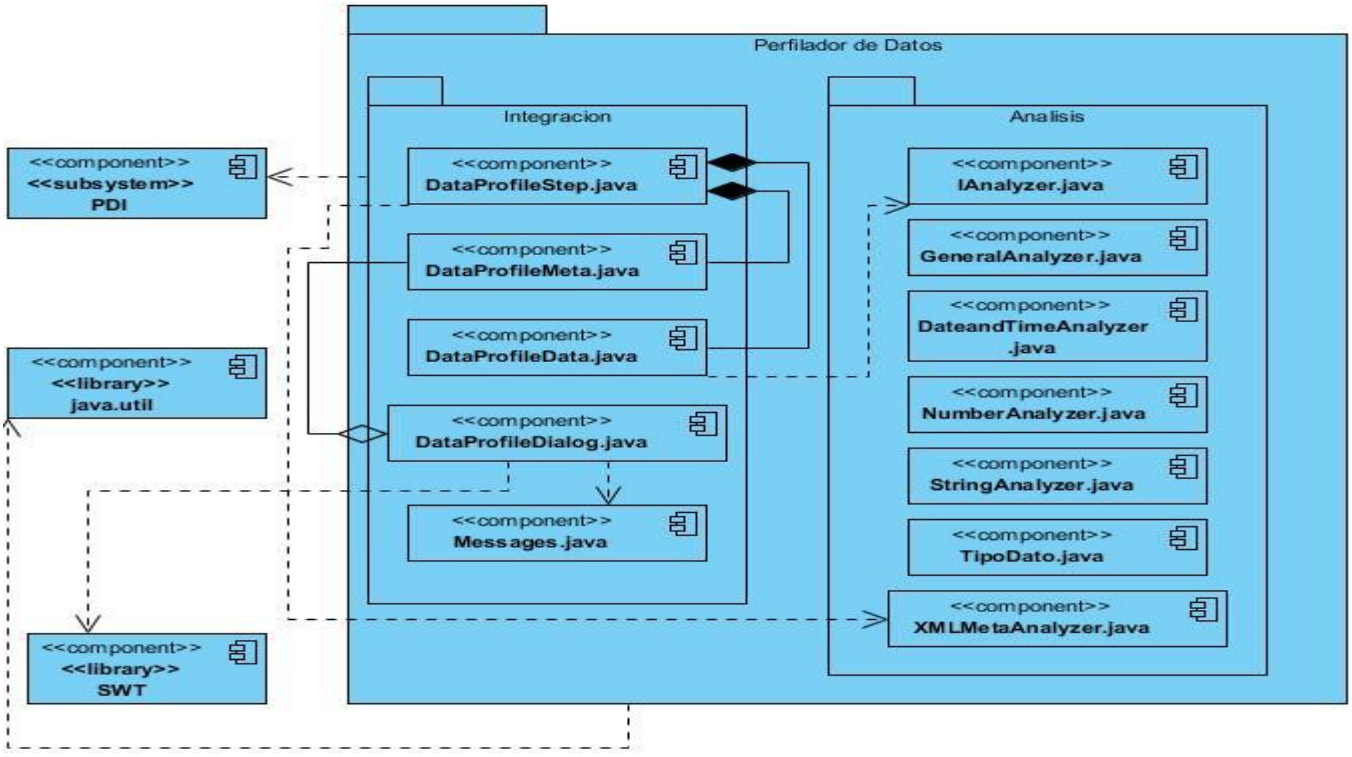



**Figura 12: Diagrama de componentes del plugin.**

### *3.2.3 Descripción de los componentes más relevantes*

A continuación se describen los componentes más importantes asociados al diseño de clases propuesto para la construcción de la extensión:

**Nombre del plugin**: “Perfilador de Datos” representa la extensión para la herramienta Pentaho Data Integration, que realiza el perfilado de datos en un flujo de trabajo a partir del tipo de análisis seleccionado.

**Subsistema PDI**: Es el subsistema que se encarga de brindar las API para el desarrollo de nuevos plugins, posee el código fuente de la aplicación (PDI) y todos los elementos relacionados con su estructura y funcionamiento.

**Paquete Integracion**: Paquete que agrupa las clases asociadas a la configuración de la extensión, las cuales son: DataProfileStep.java, DataProfileDialog.java, DataProfileMeta.java y DataProfileData.java, definidas en la estructura que propone la herramienta PDI para el desarrollo de nuevos plugins. Presenta igualmente la clase Messages.java, encargada de proporcionar los mensajes del plugin.

**Paquete Analisis**: Paquete que presenta las clases encargadas de caracterizar la configuración y llevar a cabo el trabajo de análisis sobre el flujo de datos seleccionado. Incluye las clases IAnalyzer.java, GeneralAnalyzer.java, StringAnalyzer.java, NumberAnalyzer.java, DateAndTime.java, TipoDato.java y XMLMetaAnalyzer.java.

**Library SWT**: Es una biblioteca de componentes de interfaz, dependiente de la plataforma de Pentaho. Librería encargada de establecer la configuración del plugin.
**Library java.util**: Es una biblioteca de la plataforma Java, encargada de proveer diversas clases y funcionalidades necesarias a la hora de la codificación del plugin.

**IAnalyzer.java**: Interfaz para calcular cada una de las métricas correspondientes a los tipos de análisis que realiza el plugin. Brinda la configuración de los datos a la clase DataProfileData.

**TipoDato.java**: Clase de tipo “enum” para denotar los tipos de datos que intervienen en el proceso de análisis que realiza el plugin.

**XMLMetaAnalyzer.java**: Clase encargada de brindar la estructura de los metadatos, a partir de la configuración de análisis establecida, a la clase DataProfileMeta.

**GeneralAnalyzer.java**: Clase que extiende de la interfaz IAnalyzer, implementado sus métodos abstractos a partir de las métricas correspondientes al tipo de análisis General.

**StringAnalyzer.java**: Clase que extiende de la interfaz IAnalyzer, implementado sus métodos abstractos a partir de las métricas correspondientes al tipo de análisis String.

**NumberAnalyzer.java**: Clase que extiende de la interfaz IAnalyzer, implementado sus métodos abstractos a partir de las métricas correspondientes al tipo de análisis Number.

**DateAndTimeAnalyzer.java**: Clase que extiende de la interfaz IAnalyzer, implementado sus métodos abstractos a partir de las métricas correspondientes al tipo de análisis DateTime.

**Messages.java**: Clase que utiliza la base de mensajes de la plataforma e incluye los mensajes específicos del plugin para PDI. Estos mensajes son utilizados en todos los procesos del funcionamiento del nuevo plugin.

## 3.3 Estándar de programación utilizado.

Al comenzar un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. 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. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

En el desarrollo de la extensión para la herramienta PDI sobre la plataforma Java, se pusieron en práctica algunas de las convenciones recomendadas por Sun Microsystems, que han sido difundidos y aceptados ampliamente por toda la comunidad Java, y que han terminado por consolidarse como un modelo estándar de programación. 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, salvo 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**

**Clases e interfaces:** Los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúsculas. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúsculas. 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").
**Variables:** Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas. 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 mayúsculas. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el caracter de subrayado "_".

## 3.4 Pruebas de Software

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

- Verificar la interacción de los componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente. (Pineda, 2011)

### *3.4.1 Pruebas unitarias*

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las pruebas de integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. Las pruebas unitarias pueden ser aplicadas por varias técnicas de prueba y a través de diferentes vías.

**Prueba de Caja Blanca**

La prueba de caja blanca comprueba la lógica interna del programa, para ello debe acceder al código fuente (datos y lógica) del sistema. Trabaja con entradas, salidas y el conocimiento interno.

En el caso de la presente investigación se realizan las pruebas del camino básico de manera manual por ser un software pequeño. Este tipo de pruebas tiene como característica esencial que genera un grafo de flujo a través del cual se puede calcular la complejidad ciclomática que posee la función analizada, certificando la calidad que posee el código. Este factor puede ser evaluado a través de la existencia de un rango numérico para el valor de la complejidad ciclomática que determina la evaluación de riesgo del

código en cuestión para su ejecución. El valor de la complejidad de un grafo de flujo se calcula mediante la fórmula matemática CC = A – V +2 donde:

- CC significa Complejidad Ciclomática.
- A se refiere al conjunto de arista del grafo.
- V se refiere al conjunto de vértices del grafo.

La siguiente tabla muestra el análisis de la complejidad ciclomática en un software (Martínez, 2012):

**Tabla 5: Análisis de complejidad ciclomática.**

| Complejidad ciclomática | Evaluación de riesgos |
|---|---|
| 1-10 | Programa simple, sin mucho riesgo. |
| 11-20 | Más complejo, riesgo moderado. |
| 21-50 | Complejo, programa de alto riesgo. |
| 50 en adelante. | Programa no testeable, muy alto riesgo. |

A continuación se muestran algunos ejemplos de los resultados obtenidos durante la aplicación de estas pruebas:

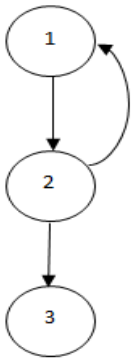



**Figura 13: Grafo de flujo del método Cálculo de varianza.**

Complejidad ciclomática del método Cálculo de varianza:

CC = A – V + 2

CC= 3 – 3 + 2

CC= 2

Evaluación de riesgo: Sin mucho riesgo.

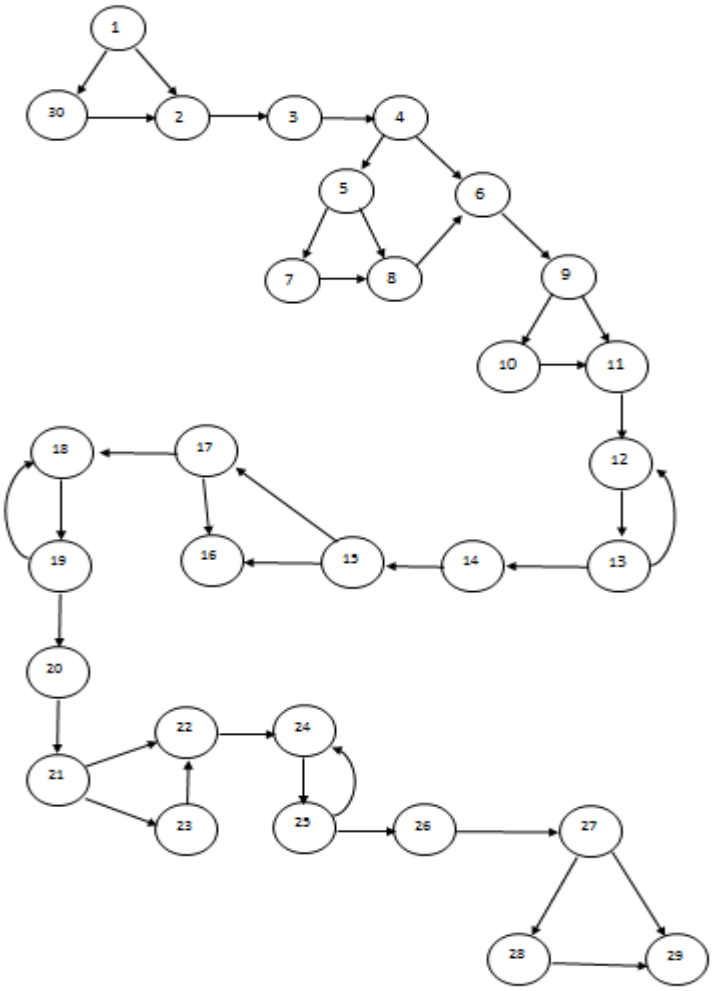



**Figura 14: Grafo de flujo del método Añadir valor (String).**

Complejidad ciclomática del método Añadir valor (String):

CC = A – V + 2

CC= 39 – 30 + 2

CC= 11

Evaluación de riesgo: Riesgo moderado.

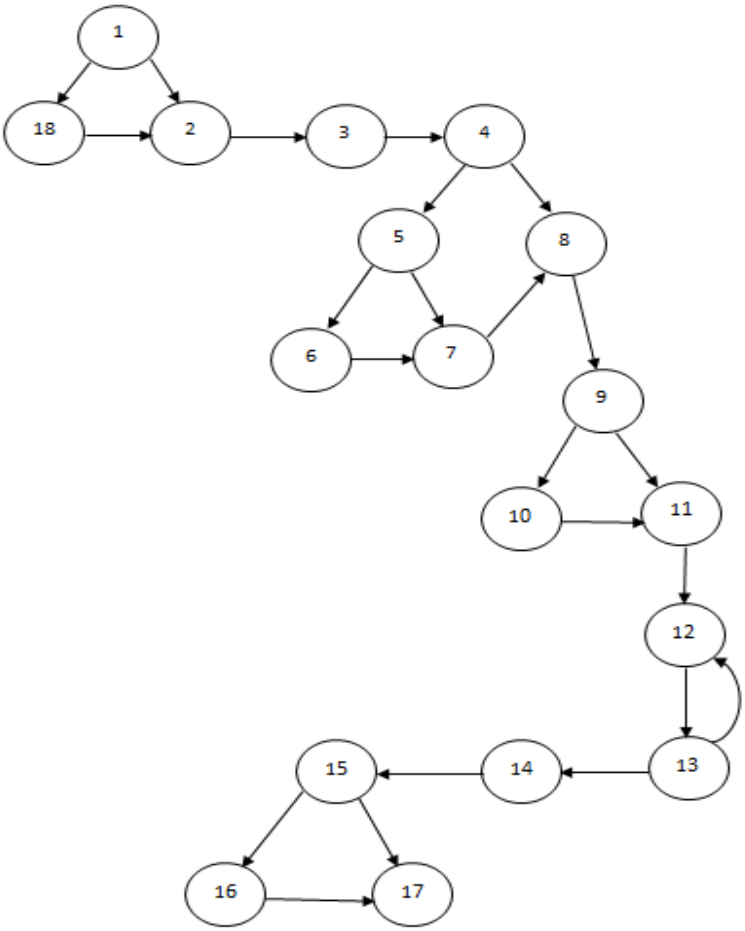



**Figura 15: Grafo de flujo del método Añadir valor (Number).**

Complejidad ciclomática del método Añadir valor (Number):

CC = A – V + 2

CC= 23 – 18 + 2

CC= 7

Evaluación de riesgo: Sin mucho riesgo.

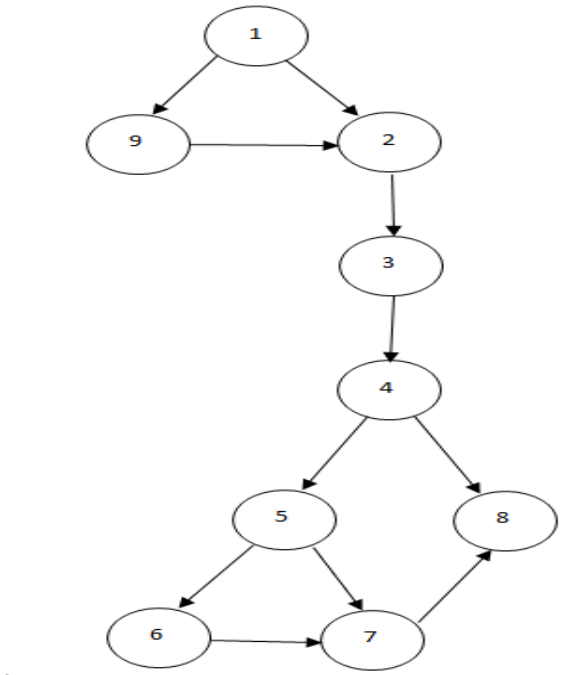



**Figura 16: Grafo de flujo del método Añadir valor (Date/Time).**

Complejidad ciclomática del método Añadir valor (Date/Time):

CC = A – V + 2

CC= 11 – 9 + 2

CC= 4

Evaluación de riesgo: Sin mucho riesgo.

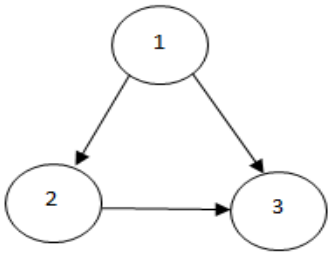



**Figura 17: Grafo de flujo del método Cálculo de media.**

Complejidad ciclomática del método Cálculo de media:

CC = A – V + 2

CC= 3 – 3 + 2

CC=2

Evaluación de riesgo: Sin mucho riesgo.

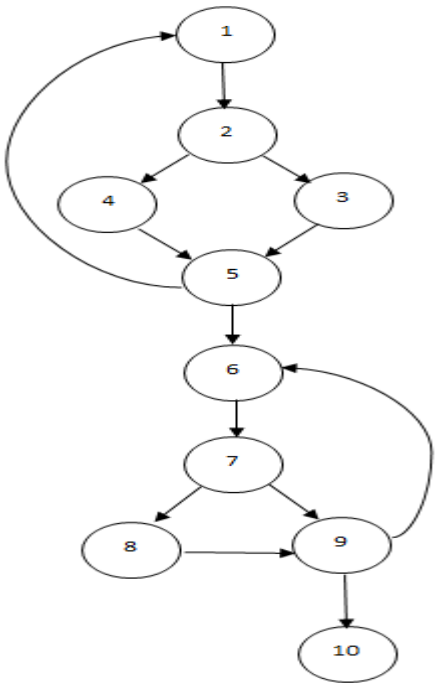



**Figura 18: Grafo de flujo del método Duplicados.**

Complejidad ciclomática del método Duplicados:

CC = A – V + 2

CC= 13 – 10 + 2

CC=5

Evaluación de riesgo: Sin mucho riesgo.

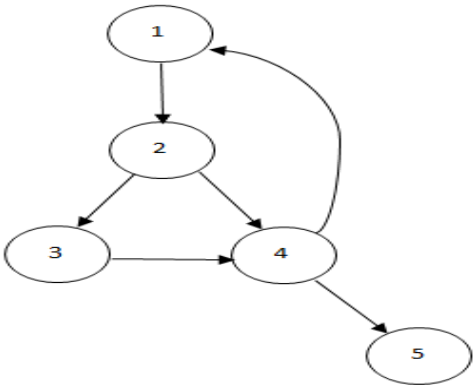



**Figura 19: Grafo de flujo del método Fecha reciente.**

Complejidad ciclomática del método Fecha reciente:

CC = A – V + 2

CC= 6 – 5 + 2

CC=3

Evaluación de riesgo: Sin mucho riesgo.

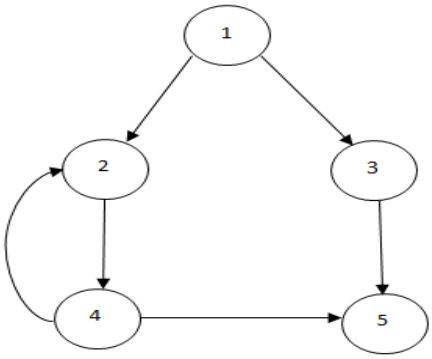



**Figura 20: Grafo de flujo del método Formato de fecha.**

Complejidad ciclomática del método Formato de fecha:

CC = A – V + 2

CC= 6 – 5 + 2

CC=3

Evaluación de riesgo: Sin mucho riesgo.

**Resultados de las pruebas unitarias**

Se analizaron 8 métodos, correspondiente a las clases o funcionalidades de mayor importancia para el desempeño del plugin para perfilado de datos, de acuerdo a lo establecido durante el análisis de requisitos. En una primera iteración resultaron en un rango de testeo posible 5 métodos, representando el 62.5% de pruebas satisfactorias, no pudiéndose testear 3 de los métodos evaluados, lo que representa el 37.5% de pruebas no satisfactorias. En una segunda iteración los códigos no testeables fueron optimizados, encontrándose el 100 % de los métodos evaluados en las categorías sin mucho riesgo o con riesgo moderado.

### *3.4.2 Pruebas funcionales*

Las pruebas funcionales fijan su atención en la validación de las funciones, métodos, servicios y casos de uso. Durante la aplicación de esta técnica se analiza cada funcionalidad implementada para verificar que se cumplan todos los requisitos establecidos y de esta forma satisfacer las necesidades planteadas. Los objetivos de este tipo de pruebas contemplan la navegación, entrada de datos, procesamiento y obtención de resultados, enfocándose en los requisitos funcionales y casos de uso.

**Pruebas de Caja Negra**

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca. Muchos autores consideran que estas pruebas permiten encontrar:

1. Funciones incorrectas o ausentes.
2. Errores de interfaz.
3. Errores en estructuras de datos o en accesos a las Bases de Datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y terminación.

Dichas pruebas se realizan a partir de una matriz de datos donde:

V: indica válido

I: indica inválido

NA: indica que no es necesario proporcionar un valor del dato, ya que es irrelevante.

En la siguiente tabla se muestran las variables V1, V2 y V3 que representan valores de entrada de datos para los casos de prueba aplicados al caso de uso Realizar la configuración del componente para Pentaho Data Integration.

**Tabla 6: Descripción de las variables correspondientes al caso de prueba para el caso de uso "Realizar la configuración del componente para Pentaho Data Integration".**

| No | Nombre del campo | Clasificación | Valor nulo | Descripción |
|---|---|---|---|---|
| **Realizar la configuración del componente para Pentaho Data Integration** | | | | |
| **V1** | Nombre del paso | Campo de texto | no | Campo que admite caracteres de cualquier tipo. Inicialmente contiene el nombre del paso, aunque puede ser modificado. |
| **V2** | Flujo de entrada | Cuadro combinado | si | Cuadro que permite seleccionar el campo del flujo de datos para su análisis. |
| **V3** | Tipo de Análisis | Cuadro combinado | si | Cuadro que permite seleccionar el tipo de análisis a realizar a partir de un campo del flujo de datos. |

A continuación se muestra el caso de prueba aplicado al caso de uso Realizar la configuración del componente para Pentaho Data Integration:

**Tabla 7: Caso de prueba aplicado al caso de uso "Realizar la configuración del componente para Pentaho Data Integration".**

| Escenario | Descripción | V1 | V2 | V3 | Respuesta del sistema | Flujo central |
|---|---|---|---|---|---|---|
| **EC-1.1 Sustituir el nombre del paso.** | En este escenario se sustituye, a gusto del usuario, el nombre del paso en la transformación. | V | NA | NA | El sistema muestra el nombre del paso cambiado. | 1-El usuario abre la interfaz del componente.<br>2-El usuario sustituye el nombre del paso.<br>3-El usuario selecciona la opción Vale.<br>4-El sistema guarda el nuevo nombre del paso. |
| **EC-1.2 Seleccionar el campo del flujo de datos a analizar.** | En este escenario se selecciona el campo del flujo de datos para realizarle el perfilado de datos a partir de un tipo de análisis. | NA | V | NA | El sistema muestra el campo seleccionado. | 1-El usuario abre la interfaz del componente.<br>2-El usuario selecciona el campo del flujo de datos que desea analizar.<br>3-El usuario selecciona la opción Vale.<br>4-El sistema guarda el campo seleccionado. |

| | | | | | | |
|---|---|---|---|---|---|---|
| **EC-1.3 Seleccionar el tipo de análisis a realizar al campo del flujo de datos.** | En este escenario se selecciona el tipo de análisis para determinar perfilado de datos a realizar sobre el campo del flujo de datos. | NA | NA | V | El sistema muestra el tipo de análisis seleccionado. | 1-El usuario abre la interfaz del componente.<br>2-El usuario selecciona el tipo de análisis que desea realizar sobre el campo del flujo de datos.<br>3-El usuario selecciona la opción Vale.<br>4-El sistema guarda el tipo de análisis seleccionado. |
| **EC-1.4 Realizar el caso de uso con datos correctos.** | En este escenario se cambia el nombre del paso, se selecciona el campo del flujo de datos para el análisis y es elegido un tipo de análisis para el perfilado de datos. | V | V | V | El sistema muestra el resultado de la acción realizada. | 1-El usuario abre la interfaz del componente.<br>2-El usuario sustituye el nombre del paso.<br>3-El usuario selecciona el campo del flujo de datos que desea analizar.<br>4-El usuario selecciona el tipo de análisis que desea realizar sobre el campo del flujo de datos.<br>5-El usuario selecciona la opción Vale.<br>5-El sistema guarda la configuración correcta del componente. |

| | | | | | | |
|---|---|---|---|---|---|---|
| **EC-1.5 Realizar el caso de uso con datos incorrectos.** | En este escenario se selecciona el tipo de análisis para determinar perfilado de datos a realizar sobre el campo del flujo de datos. | V | I | I | El sistema muestra un mensaje de error. | 1-El usuario abre la interfaz del componente.<br>2-El usuario sustituye el nombre del paso.<br>3-El usuario selecciona el campo del flujo de datos que desea analizar.<br>4-El usuario selecciona un tipo de análisis diferente al tipo de datos entrado en el campo del flujo de datos.<br>5-El usuario selecciona la opción Vale.<br>5-El sistema guarda la configuración incorrecta del componente. |

Para validar el correcto funcionamiento de la extensión fueron realizadas tres iteraciones de pruebas.

En la siguiente tabla se muestra un resumen general de todas las dificultades encontradas por iteraciones en las pruebas realizadas, donde:

PD: Pendiente

RA: Resuelta

**Tabla 8: Resumen de los resultados de las pruebas aplicadas.**

| Fecha | Versión | Caso de prueba | Cantidad de no conformidades | Cantidad de no conformidades PD | Cantidad de no conformidades RA |
|---|---|---|---|---|---|
| 19/05/2014 | 1.0 | CU Realizar la configuración del componente para Pentaho Data Integration. | 3 | - | 3 |
| 20/05/2014 | 1.1 | CU Realizar la configuración del componente para Pentaho Data Integration.<br>CU Realizar análisis general en un flujo de datos. | 1<br>2 | -<br>- | 1<br>2 |
| 21/05/2014 | 1.2 | CU Realizar la configuración del componente para Pentaho Data Integration.<br>CU Realizar análisis general en un flujo de datos. | 0<br>0 | -<br>- | 0<br>0 |

Luego de realizar las pruebas de Caja Negra en tres iteraciones de prueba, mediante los casos de prueba asociados a cada caso de uso, se comprobó que el plugin da cumplimiento de forma correcta a todos los requisitos funcionales definidos. Fueron validadas las entradas de datos en la interfaz de usuario, para

que la generación de los diagramas del análisis y del código fuente se realice solo con datos correctos. Se comprobó que la aplicación construida a partir de la generación de código fuente funcione de forma correcta.

### *3.4.3 Pruebas de integración*

Las Pruebas de Integración son aquellas que permiten probar en conjunto distintos subsistemas funcionales o componentes del proyecto para verificar que interactúan de manera correcta y que se ajustan a los requisitos especificados (sean estos funcionales o no).

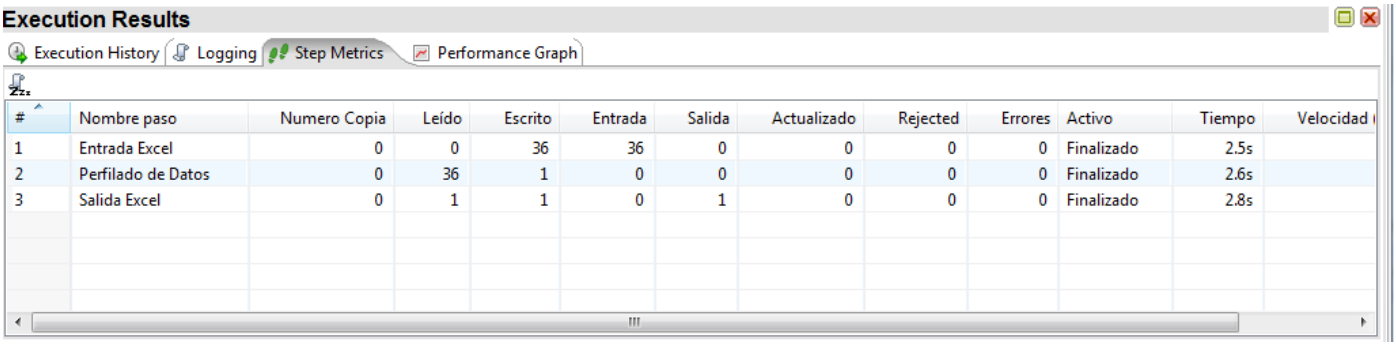
Este tipo de pruebas deberán ejecutarse una vez se haya asegurado el funcionamiento correcto de cada una de las funcionalidades por separado, es decir, una vez se hayan ejecutado sin errores las pruebas unitarias al plugin. Posteriormente se procederá a integrar dichas funcionalidades y se añadirá el plugin con el resto de los componentes del PDI, siguiendo una estrategia incremental ascendente. A medida que se integren las funcionalidades, se aplicarán pruebas funcionales para favorecer la detección de errores y no conformidades.

La aplicación de este tipo de pruebas en un proyecto proporciona una serie de ventajas, especialmente relacionadas con la prevención de la aparición de errores ocasionados por:

- Un mal diseño de los interfaces de los componentes
- Un mal uso de los componentes

Además, al igual que las pruebas unitarias, las pruebas de integración sirven como documentación del uso de los componentes puesto que en definitiva, de nuevo podrán considerarse como ejemplos de su uso. (Alfaro Medina, y otros, 2013)


**Figura 21: Componente para perfilado de datos en funcionamiento a partir de una transformación simple.**




| # | Nombre paso | Numero Copia | Leído | Escrito | Entrada | Salida | Actualizado | Rejected | Errores | Activo | Tiempo | Velocidad |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Entrada Excel | 0 | 0 | 36 | 36 | 0 | 0 | 0 | 0 | Finalizado | 2.5s | |
| 2 | Perfilado de Datos | 0 | 36 | 1 | 0 | 0 | 0 | 0 | 0 | Finalizado | 2.6s | |
| 3 | Salida Excel | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | Finalizado | 2.8s | |

**Figura 22: Datos de análisis arrojados a partir de la ejecución del componente para perfilado de datos.**

Luego de ejecutadas las pruebas de integración, el plugin para perfilado de datos presentó una aceptación del 100% por parte del sistema, posibilitando su inclusión en el mismo en cualquier momento que sea necesario.

### *3.4.4 Pruebas de aceptación*

Dentro del tipo de pruebas que existen para validar el software, una de las más importantes son las de aceptación (user's tests). Son aquellas pruebas que son diseñadas por el propio equipo de desarrollo en base a los requisitos funcionales especificados en la fase de análisis para cubrir todo ese espectro, y ejecutadas por el propio usuario final, no por todos evidentemente, pero sí por una cantidad de usuarios finales significativo que den validez y conformidad al producto que se les está entregado en base a lo que se acordó inicialmente.

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. (Alfaro Medina, y otros, 2013)

Las pruebas de aceptación fueron realizadas por el cliente, quedando el mismo satisfecho con la solución propuesta, a partir de los resultados arrojados por el conjunto de pruebas trazadas y de la familiarización con el trabajo y la interfaz del plugin.

## Conclusiones parciales

La descripción de las consideraciones que se deben tener en cuenta para la implementación de extensiones para la herramienta PDI permitió un mayor entendimiento, facilitando el proceso de desarrollo e integración de la extensión. A partir de un análisis realizado acerca de los componentes fundamentales asociados al diseño de clases propuesto para la construcción del plugin, fueron identificados y descritos 15 componentes que conforman el diagrama de componentes del sistema. Para la implementación de la extensión propuesta se consideró necesario aplicar algunos estándares de programación para garantizar que la codificación se realice correctamente.

# Conclusiones

Como resultado del presente trabajo de diploma y dando cumplimiento a los objetivos propuestos, se puede establecer como conclusiones las siguientes:

- A partir del estudio de los elementos teóricos para el desarrollo de aplicaciones de gestión de información, se determinó que la metodología Open Up fue la más idónea para ser aplicada en el desarrollo de la solución.
- Se analizaron los aspectos fundamentales sobre el desarrollo de plugin para la herramienta Pentaho Data Integration facilitando la implementación del plugin.
- Teniendo en cuenta las exigencias y necesidades del centro DATEC relacionadas con el proceso de ETL, fueron identificados e implementados los requisitos de la extensión, obteniéndose como resultado 21 requisitos funcionales que fueron agrupados en cinco casos de uso. La aplicación de los patrones GRASP permitió el diseño adecuado y la correcta asignación de responsabilidades a cada una de las clases implementadas. De ese modo se obtuvo un componente de perfilado de datos para PDI capaz de realizar los análisis principales que se realizan en el centro.

# Recomendaciones

Se recomienda para posteriores versiones del plugin la implementación de nuevos tipos de análisis con sus respectivas funcionalidades.

Seguir profundizando en el trabajo de perfilado de datos para añadir nuevas funcionalidades al plugin en dependencia de las necesidades que surjan en el departamento de almacenes de datos.

# Referencias Bibliográficas


- Alvarez Marañón, Gonzalo. 2009. ¿Qué es Java? [En línea] 2009. [Citado el: 12 de Octubre de 2013.] http://www.iec.csic.es/CRIPTONOMICON/java/funcionamiento.html.
- Fernández, Carlos. 2009. DataPrix. [En línea] 17 de Agosto de 2009. [Citado el: 12 de Marzo de 2014.] http://www.dataprix.com/data-profiling-sql-server-2008.
- Chodnicki, Slawomir. 2010. Adventures with Open Source BI. [En línea] 13 de Junio de 2010. [Citado el: 29 de Abril de 2014.] http://type-exit.org/adventures-with-open-source-bi/2010/06/developing-a-custom-kettle-plugin-a-simple-transformation-step/.
- Diaz Ordaz, Jesús. 2013. Gravitar. [En línea] 10 de Noviembre de 2013. [Citado el: 4 de Diciembre de 2013.] http://www.gravitar.biz/index.php/herramientas-bi/pentaho/caracteristicas-pentaho.
- Espinosa, Roberto. 2010. El Rincón del BI. [En línea] 10 de Mayo de 2010. [Citado el: 15 de Enero de 2014.] http://churriwifi.wordpress.com/category/etl/page/2/.
- —. 2010. El Rincón del BI. [En línea] 1 de Junio de 2010. [Citado el: 20 de Enero de 2014.] http://churriwifi.wordpress.com/2010/06/01/comparativa-talend-vs-kettle-pdi/.
- Mejia, Orlando JC. 2012. Calaméo. [En línea] 4 de Junio de 2012. [Citado el: 10 de Diciembre de 2013.] http://www.calameo.com/books/00145501612d9387159b4.
- Microsoft. 2014. Microsoft Developer Network. [En línea] 2014. [Citado el: 30 de Marzo de 2014.] http://msdn.microsoft.com/es-es/library/dd409390.aspx.
- Olguin Juarez, Efrain Alberto. 2013. OpenUP. [En línea] 5 de Septiembre de 2013. http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.
- Pentaho Solutions. 2013. Pentaho Business Intelligence. [En línea] 29 de Octubre de 2013. [Citado el: 25 de Noviembre de 2013.] https://sites.google.com/site/pentahobisuite/home/caracteristicas/pentaho-dashboards/pentaho-data-integration.
- Soto, David. 2008. Integracion y Calidad de Datos. [En línea] 17 de Julio de 2008. [Citado el: 21 de Enero de 2013.] http://integracionycalidad.blogspot.com/2008_07_01_archive.html.

- The Data Integration Company. 2011. Informática: The Data Integration Company. [En línea] 2011. [Citado el: 6 de Febrero de 2014.] http://origin-wwwnew.informatica.com/es/products_services/data_explorer/Pages/index.aspx.
- Vidal Gil, Juan. 2011. DataPrix. [En línea] 3 de Septiembre de 2011. [Citado el: 19 de Febrero de 2014.] http://www.dataprix.com/blogs/juan-vidal/le-estamos-dando-importancia-que-se-merecen-procesos-calidad-datos.
- Visual Paradigm Company. 2013. Visual Paradigm. [En línea] 2013. [Citado el: 13 de Diciembre de 2013.] http://www.visual-paradigm.com/.

# Bibliografía


- Acuña, Karenny Brito. 2009. Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad deInformática de Cienfuegos. Cienfuegos : s.n., 2009. págs. 34-36.
- Alfaro Medina, Jazmín, y otros. 2013. Pruebas de Software y Pruebas de Aceptación. 2013.
- Johannes Mayer, I.M. 2006. Ligthweight Plug-in Based Aplication Development. 2006.
- Márquez, José, y otros. 2012. Integración de Java y Prolog. Mérida : s.n., 2012. pág. 2.
- Amaro Calderón, Sarah Dámaris y Valverde Rebaza, Jorge Carlos. 2007. Metodologías Ágiles. Trujillo : s.n., 2007. págs. 6-18.
- Ariza Rojas, Maribel y Molina García, Juan Carlos. 2004. Introducción y principios básicos del Desarrollo de Software basado en Componentes. Universidad Javeriana. Bogotá : s.n., 2004. págs. 1-2.
- Astudillo, Hernán y Visconti, Marcello. Fundamentos de la Ingeniería de Software.
- Reynoso, Carlos. 2005. Introducción a la Arquitectura de Software. Buenos Aires : s.n., 2005.
- EVA. 2007-2008. Conferencia #3. Flujo de trabajo de Requerimientos. Ingeniería de Software, UCI. La Habana : s.n., 2007-2008. Presentación de Power Point.
- Hernández, Cinolkis Cobas. 2010. Desarrollo del Componente de Seguridad para el subsistema Mondrian perteneciente al sistema Pentaho. Universidad de las Ciencias Informáticas. La Habana : s.n., 2010. pág. 24.
- Métodos "I + D" de la Informática. Barchini, Graciela Elisa. 2005. 5, Santiago de Estero : s.n., 2005, Revista de Informática Educativa y Medios Audiovisuales, Vol. II, págs. 16-24.
- Otero Vidal, Mari Carmen. 2008. Ingeniería de Software.UML Cap 4. 2008.
- Mínguez Porras, Alejandro. 2011. Fundamentos de Calidad de Datos. 2011. págs. 28-34, Presentación.

- Martínez, Ing. Eduardo Salazar. 2012. Propuesta de procedimiento para realizar pruebas de Caja Blanca a las aplicaciones que se desarrollan en el lenguaje Python. Cuba: Facultad regional de Granma., 2012.