

**Universidad de las Ciencias Informáticas**

**Facultad 3**



**Herramienta para la generación de datos en los  
subsistemas "Contabilidad" y "Costos y Procesos".**

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

**Autores:**

José Luis Bizet Romero.

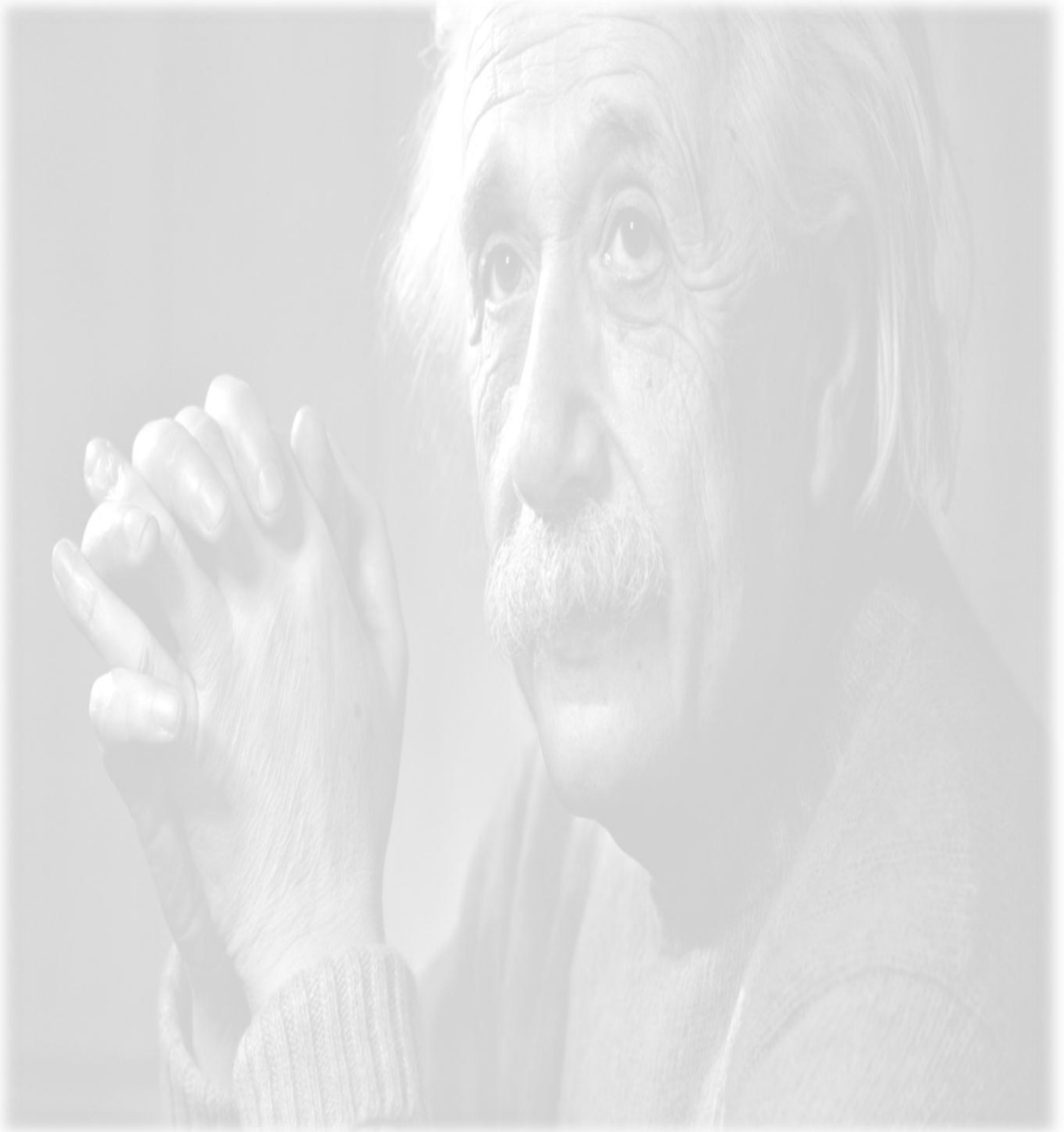
Reinier González Medinilla.

**Tutor:**

Ing. Alain Fernández Deronceré

***"Ciudad de La Habana. Julio, 2014"***

***"Año 55 de la Revolución"***



*"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad".*

*Albert Einstein*

## DECLARACIÓN DE AUTORÍA

Declaro que somos los únicos autores del presente trabajo que lleva como título: Herramienta para la Generación de Datos en los subsistemas “Contabilidad y Costos y Procesos” y autorizamos al Centro para la Informatización de Gestión de Entidades de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

José Luis Bizet Romero

---

**Firma del Autor**

Reinier González Medinilla

---

**Firma del Autor**

Ing. Alain Fernández Deronceré

---

**Firma del Tutor**

### DATOS DE CONTACTO

#### Síntesis del tutor:

Ing. Alain Fernández Deronceré

Graduado de Ingeniero en Informáticas en la Universidad de Holguín. Con 8 años de experiencia en el desarrollo de software en el CEIGE.

Correo electrónico: [afernandezd@uci.cu](mailto:afernandezd@uci.cu)

### AGRADECIMIENTOS

*Primero que todo quiero agradecer a la persona más importante y que más quiero en mi vida: mi mamá, porque gracias a ella hoy he llegado hasta aquí.*

*A mi papá por darme todo su apoyo y levantarme el ánimo cuando más lo necesitaba.*

*A mis hermanos que han sido fuentes de inspiración para mí y más que todo un ejemplo a seguir.*

*A mi familia por su apoyo y por de una forma u otra influir en este logro, especialmente a mis tíos Yamila, Pepincito, Maritza, Cari, Hipólito, Ricel, a mis primos Yafer, Adrián, Viviana, Eduardito, Julito, Taimí.*

*A mis abuelos que aunque ya no estén físicamente conmigo, seguro que están orgullosos de su nieto. Y siempre van a estar en mi corazón.*

*A mis amigos del barrio que siempre me ayudaron desde el momento que llegue a la UCA. En especial a Flavio que ha sido un hermano para mí y siempre ha estado para ayudarme en todo momento. Y parte de este logro se lo agradezco a él.*

*A mi tática por haberme dado todo su amor incondicional, por entenderme estos últimos meses y haberme dado el apoyo que necesitaba para poder salir de los malos momentos. Te Amo Tática.*

*A los amigos que hice en la Universidad de los cuales nunca me voy a olvidar. Gracias a ellos por estar conmigo en todos momentos a Yaniel, Miguel, Artime, Alonso, Osbel, Sobrino, Pablo, Andrew, Dian, Roberto, Eiler.*

*A mi compañero de tesis con el cual pudimos sacar este trabajo hacia adelante. Con mucho esfuerzo y dedicación.*

*José Luis Bizet Romero*

*En especial a mis abuelos, a quien les debo toda mi vida, les agradezco su cariño y comprensión, a ellos quienes han sabido formarme con buenos sentimientos, hábitos y valores.*

*A mis padres por traerme al mundo, ayudarme y mostrarme confianza en todo momento los quiero mucho.*

*A mi hermano Dainier por ser parte de las personas más importantes de mi vida y apoyarme en todo momento, eres el mejor hermano del mundo... TE QUIERO*

*A mi tío Ricardo Medinilla el cual me ha enseñado que la familia es lo más importante en la vida.*

*A mis amigos que he hecho en esta universidad en especial a Gaby, Julio, Eiler, el Rafa, Eddy, Damaris, Hurshel, Dachel (el crack), Carlos Javier, Ledian, Sanamé, Orlando (el cuadro), Osmail, Abelito, Alejandro, Juan Pablo, Yoelvis (el yipi) Rene, Zoemi, Zoima, Jorge Jesús, Carlos Llama, Noel, Sedeño por haberse convertido en parte de mi familia.*

*A dos amigos especiales a los cuales considero mis hermanos Andrew y Paulito gracias por estar siempre a mi lado en los momentos bueno y malos a lo largo de estos 5 años de universidad.*

*A mis co-tutores Ander, Vismar, Flavio los cuales me apoyaron mucho para lograr terminar en tiempo y con calidad este trabajo de diploma.*

*A mi compañero de tesis José Luis, pues lo considero más que un amigo un hermano que ha sabido ganarse mi respeto y mi apoyo incondicional en todo momento.*

DEDICATORIA

*A mis abuelos que aunque ya no están aquí siempre estarán en mi corazón y formarán parte de mi vida.*

*A mi mamá y mi papá por todo su amor y por haberme impulsado a seguir aun cuando no tenía fuerzas, por ser mis guías y mi razón de ser y sin ellos ahora no estaría aquí.*

*José Luis Bizet Romero*

*Este trabajo está dedicado especialmente a mis abuelos, por ser los principales impulsores de mi formación, por confiar en mí y sentirse tan orgullosos del nieto que tienen. ``Este título es de ustedes``.*

*Reinier González Medinilla*

### RESUMEN

En la Universidad de las Ciencias Informáticas se desarrolla el Sistema Xedro-ERP. El mismo está dividido en varios subsistemas, considerándose críticos los de “Contabilidad y Costos y Procesos”, por su naturaleza transversal, que propicia una alta dependencia de ellos para el funcionamiento del resto. Para agilizar los procesos de desarrollo y prueba en estos o cualquiera de los restantes subsistemas de Xedro-ERP y facilitar la presentación del software a clientes, se requiere sustituir el lento y engorroso mecanismo actual de carga y generación de datos.

Por esta razón el presente trabajo propone el desarrollo de una herramienta generadora de datos, que constituye un elemento de alta importancia en el proceso de desarrollo de software, ya que permite poblar las bases de datos de determinado sistema con información vital para su funcionamiento, de esta forma se agiliza y al mismo tiempo se hace más eficiente el proceso de carga y generación de datos asociados a los subsistemas de “Contabilidad y Costos y Procesos” de Xedro-ERP, ya sea mediante la creación de datos aleatorios o mediante la consulta a fuentes de datos ya existentes.

El documento recoge los resultados de la investigación realizada, describiéndose las principales características de los sistemas analizados. En el mismo se explica la arquitectura y el diseño del sistema propuesto, se describen las herramientas y tecnologías que se utilizan, así como los artefactos que se generan en el proceso de desarrollo ágil aplicado con la metodología XP.

**Palabras clave:** base de datos, carga de datos, contabilidad y costos y procesos, generación de datos.



## ÍNDICE

INTRODUCCIÓN .....	1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA .....	6
1.1    Introducción .....	6
1.2    Conceptos asociados a la investigación .....	6
1.2.1    Datos .....	6
1.2.2    Bases de datos .....	6
1.2.3    Sistemas Gestores de Base de Datos .....	7
1.2.4    Sistema Gestor de Base de Datos PostgreSQL .....	7
1.2.5    Generación de datos .....	8
1.3    Herramientas de generación de datos .....	9
1.3.1    EMS Data Generator for PostgreSQL .....	9
1.3.2    Datanamic Data GeneratorMultiDB .....	10
1.3.3    Postgen.....	10
1.3.4    Comparación entre las herramientas analizadas.....	10
1.4    Metodología de desarrollo .....	12
1.4.1    Extreme Programming (XP) .....	13
1.5    Herramientas, lenguajes y tecnologías utilizadas.....	15
1.5.1    Lenguaje de modelado.....	16
1.5.2    Herramienta CASE.....	16
1.5.2.1    Visual Paradigm .....	17
1.5.3    Lenguaje de programación.....	17
1.5.4    Lenguaje Java.....	17
1.5.5    Entorno de desarrollo integrado .....	18
1.5.5.1    NetBeans.....	18
1.6    Patrones de diseño.....	19
1.7    Métricas utilizadas .....	20
1.8    Pruebas de calidad .....	22
1.8.1    Justificación de las pruebas a utilizar .....	22
1.9    Conclusiones parciales.....	24
CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA .....	25
2.1    Introducción .....	25
2.2    Propuesta del sistema .....	25
2.3    Historias de usuarios .....	25

2.3.1	Lista de reserva de productos .....	27
2.3.2	Plan de Iteraciones .....	30
2.3.3	Tareas de ingeniería .....	31
2.4	Arquitectura de software .....	32
2.4.1	Estilos arquitectónicos.....	32
2.4.2	Arquitectura en capas .....	33
2.5	Patrones de diseño.....	34
2.5.1	Patrones GRASP .....	34
2.5.2	Patrones GoF.....	38
2.6	Diseño de la solución.....	40
2.6.1	Tarjetas CRC .....	40
2.6.2	Diagrama de clases .....	41
2.7	Conclusiones parciales.....	41
CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN .....		43
3.1	Introducción .....	43
3.2	Implementación .....	43
3.2.1	Estándares de codificación .....	43
3.2.2	Principales bibliotecas utilizadas .....	44
3.3	Interfaces de usuario .....	46
3.4	Validación de requisitos .....	47
3.5	Verificación del diseño.....	48
3.5.1	Tamaño Operacional de Clase (TOC).....	48
3.5.2	Relaciones entre Clases (RC).....	50
3.6	Pruebas software.....	54
3.6.1	Pruebas unitarias .....	54
3.6.2	Pruebas de aceptación.....	56
3.6.3	Resultados de pruebas .....	58
3.6.4	Pruebas de usabilidad.....	59
3.7	Validación de la variable de la investigación.....	60
3.8	Conclusiones parciales.....	61
CONCLUSIONES GENERALES .....		62
RECOMENDACIONES .....		63
BIBLIOGRAFÍA REFERENCIADA.....		64

### ÍNDICE DE FIGURAS

Figura 1 Arquitectura en capas.....	33
Figura 2 Fragmento del diagrama de clases donde se evidencia el patrón experto.....	35
Figura 3 Fragmento del diagrama de clases donde se evidencia el patrón alta cohesión.....	36
Figura 4 Fragmento del diagrama de clases donde se evidencia el patrón bajo acoplamiento.....	36
Figura 5 Fragmento del diagrama de clases donde se evidencia el patrón creador.....	37
Figura 6 Fragmento del diagrama de clases donde se evidencia el patrón controlador.....	38
Figura 7 Fragmento del diagrama de clases donde se evidencia el patrón observador.....	39
Figura 8 Fragmento del diagrama de clases donde se evidencia el patrón estrategia.....	39
Figura 9 Diagrama de clases.....	41
Figura 10 Fragmento de código donde se evidencia la biblioteca JDBC.....	45
Figura 11 Fragmento de código donde se evidencia el uso la biblioteca xeger.....	46
Figura 12 Interfaz de usuario #1 Crear proyecto.....	46
Figura 13 Interfaz de usuario #2 Crear árbol de dependencias.....	47
Figura 14 Resultados de la métrica TOC. Atributo: Responsabilidad.....	49
Figura 15 Resultados de la métrica TOC. Atributo: Complejidad.....	49
Figura 16 Resultados de la métrica TOC. Atributo: Reutilización.....	50
Figura 17 Resultados de la métrica RC. Atributo: Acoplamiento.....	52
Figura 18 Resultados de la métrica RC. Atributo: Mantenimiento.....	53
Figura 19 Resultados de la métrica RC. Atributo: Cantidad de pruebas.....	53
Figura 20 Resultados de la métrica RC. Atributo: Reutilización.....	53
Figura 21 Pruebas unitarias #1.....	55
Figura 23 Diseño de clases de prueba para la historia de usuario Generar datos mediante un formato.....	58
Figura 24 Resultados de prueba.....	59
Figura 25 Principios de la heurística de usabilidad.....	60

### ÍNDICE DE TABLAS

Tabla 1 Comparación entre las herramientas analizadas. ....	12
Tabla 2 Umbrales de la métrica Tamaño Operacional de Clase. ....	21
Tabla 3 Umbrales de la métrica Relaciones entre clases ....	22
Tabla 4 Historia de usuario Generar datos mediante un formato. ....	26
Tabla 5 Historia de usuario generar datos mediante una lista. ....	27
Tabla 6 Lista de reservas de productos. ....	30
Tabla 7 Plan de iteración. ....	31
Tabla 8 Tarea de ingeniería Generar datos mediante un formato. ....	32
Tabla 9 Tarea de ingeniería Generar datos mediante una lista ....	32
Tabla 10 Tarjeta CRC de la clase proyecto. ....	40
Tabla 11 Clases a las que se les aplicó la métrica TOC. ....	49
Tabla 12 Clases del diagrama del diseño a las que se les aplico la métrica RC. ....	52

## INTRODUCCIÓN

En la actualidad existe una tendencia al perfeccionamiento de las operaciones y actividades empresariales haciendo uso de las Tecnologías de la Información y las Comunicaciones (TIC). La gran mayoría de las soluciones para la gestión de empresas y otras entidades de servicio público, generan grandes volúmenes de información asociada al negocio que manejan. Existe entonces, la necesidad de que esta información no solo sea almacenable, sino que pueda ser consultada y visualizada en distintos formatos para apoyar así, la toma de decisiones de la organización.

La mayoría de los autores coinciden en que una de las principales ventajas de informatizar los procesos empresariales la determina el hecho de renunciar al procesamiento manual de estos grandes volúmenes de datos, que en la mayor parte de los casos constituye un proceso lento y engorroso. Por lo anteriormente explicado, uno de los principales campos de estudio de la informática a lo largo de la historia ha estado dirigido a propiciar el desarrollo de tecnologías que faciliten el manejo de los datos, como consecuencia surgen las bases de datos (BD). Aparejado al avance de las tecnologías de BD han sido desarrollados sistemas gestores de bases de datos (SGBD), estos sistemas permiten manejar y controlar la información, así como una mejor gestión de los datos (Pérez, 2010).

Un punto crítico en el proceso de desarrollo de aplicaciones para la gestión empresarial lo constituye la realización de pruebas al software, buscando simular un entorno acorde al ambiente real donde finalmente será explotada la aplicación en cuestión (Korel, 1990). Normalmente para que los desarrolladores puedan realizar estas pruebas, se necesitan poblar las bases de datos con determinados volúmenes de información, indispensables para probar las funcionalidades del software. Crear dicha información manualmente implica un gasto de tiempo y esfuerzo considerable. Las dificultades antes mencionadas fueron resueltas en gran medida con el surgimiento de las herramientas generadoras de datos, las cuales tienen como objetivo poblar las BD, no solo con el fin de ser usadas para el proceso de prueba, sino también cuando se presenta la aplicación al cliente.

Cuba en los últimos años ha registrado un crecimiento considerable en el desarrollo del software. Es por ello que el gobierno tiene dentro de sus principales retos, impulsar la industria cubana del software con el objetivo de informatizar la sociedad y encontrar mercado en diversos destinos del mundo. Entre las instituciones que se dedican al desarrollo de soluciones informáticas en Cuba se encuentra la Universidad de las Ciencias Informáticas (UCI).

La UCI en correspondencia con las necesidades del país, organiza su trabajo por áreas de desarrollo, contando con un conjunto de centros productivos entre los cuales se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE). Este centro enfoca sus esfuerzos en el desarrollo de proyectos encargados de informatizar las áreas de procesos de una entidad o empresa, tanto nacional como internacional. Uno de los principales proyectos llevados por CEIGE es el ERP-Cuba cuyo fin es el desarrollo del Sistema de Gestión Integral para Entidades Xedro-ERP.

En el caso de Xedro-ERP está compuesto por un conjunto de subsistemas entre los cuales se consideran críticos los subsistemas de “Contabilidad y Costos y Procesos”, ya que su funcionamiento es transversal para los demás módulos del sistema al ser bases desde el punto de vista contable, existiendo así una alta dependencia de ellos para el funcionamiento del resto.

Uno de los problemas más comunes que enfrentan los equipos de desarrollo y prueba dentro del proyecto Xedro-ERP es la no existencia de juegos de datos para simular correctamente los procesos reales tanto en los subsistemas “Contabilidad y Costos y Procesos”, como en aquellos que dependan de estos dos para su funcionamiento. Actualmente la carga de datos reales para procesos de prueba y desarrollo se realizan a través de scripts compuestos por sentencias del Lenguaje de Manipulación de Datos (por sus siglas del inglés DML) y Lenguaje de Definición de Datos (por sus siglas del inglés DDL). Otra forma común es la utilización de una hoja de cálculo Excel, exportada al formato Valores Separados por Comas (por sus siglas del inglés CSV) y posteriormente importada mediante una herramienta.

Estos procesos además de ser lentos y engorrosos por la gran cantidad de tablas y relaciones que existen en estos subsistemas, tienen otra complicación importante: los datos deben cumplir con una semántica dentro del negocio implícito en el sistema, es decir que deben organizarse de tal manera que puedan interpretarse de manera significativa sin la intervención humana. La semántica de datos tiene como objetivo representar el mundo real con la mayor precisión posible, dentro del conjunto de datos. Los símbolos de datos se organizan linealmente y jerárquicamente para dar ciertos significados como el descrito anteriormente (Semagix, 2009); lograr una correcta semántica en los datos es una de las tareas más complicadas durante el proceso de generación.

Las dificultades mencionadas anteriormente para poblar las bases de datos correspondientes a los subsistemas de “Contabilidad y Costos y Procesos”, acarrearán diversas consecuencias. La primera y más importante es la limitación para realizar pruebas de calidad al software en estos subsistemas o en cualquier otro que dependa de ellos. De igual manera se dificulta hacer presentaciones del software a clientes potenciales interesados en adquirir el sistema Xedro-ERP. Por último, el sistema y la

organización en la que se desee implantar, requieren cierta información de nomencladores o histórica que debe ser cargada previamente antes de iniciar la explotación del software, algo que mediante una de las técnicas usadas actualmente puede tornarse sumamente lento.

Teniendo en cuenta los elementos anteriormente planteados se define como **problema a resolver**: la forma en que se realiza la carga de datos en los subsistemas base de Xedro-ERP, limita la disponibilidad de los datos para la realización de actividades de prueba y la presentación a potenciales clientes.

**Objeto de estudio**: herramientas de generación de datos, enmarcado en el **campo de acción**: procesos de generación de datos para bases de datos relacionales.

Se identifica como **objetivo general**: desarrollar una herramienta para la generación de datos en los subsistemas “Contabilidad y Costos y Procesos” de Xedro-ERP, de manera que contribuya a la oportuna disponibilidad de datos requeridos para la preparación de actividades de prueba y presentación a potenciales clientes.

A partir del objetivo general se definen los siguientes **objetivos específicos**:

- ✓ Realizar la fundamentación teórica de la investigación relativa a los sistemas de generación de datos.
- ✓ Realizar el análisis y diseño de la herramienta a desarrollar.
- ✓ Realizar la implementación de la herramienta.
- ✓ Validar la solución obtenida.

Proponiéndose como **idea a defender**: con el desarrollo de la herramienta de generación de datos para los subsistemas “Contabilidad y Costos y Procesos” se mejora la disponibilidad de los datos para la realización de actividades de prueba y la presentación a potenciales clientes.

Con el fin de dar cumplimiento a los objetivos especificados anteriormente se realizarán las siguientes **tareas de investigación**:

- ✓ Análisis de las principales herramientas existentes para la generación de datos en PostgreSQL.
- ✓ Fundamentación sobre las herramientas, metodología, lenguajes y tecnologías que se usará para el desarrollo.
- ✓ Identificación de las funcionalidades de la herramienta.

- ✓ Elaboración de las Historias de Usuario de la herramienta a desarrollar.
- ✓ Elaboración de las Tarjetas Clases-Responsabilidad-Colaboración de la herramienta.
- ✓ Realización del diagrama de clases del diseño de la herramienta a desarrollar.
- ✓ Validación de los artefactos generados en el diseño.
- ✓ Implementación de la herramienta.
- ✓ Realización de pruebas unitarias a la herramienta desarrollada.
- ✓ Realización de pruebas de aceptación a la herramienta desarrollada.
- ✓ Validación de las variables de la investigación.

Los **métodos** que sustentan la investigación son:

### **Métodos teóricos:**

- **Histórico-lógico:** permitió consultar información de diferentes autores para obtener el conocimiento de los antecedentes, la evolución y desarrollo que han tenido las herramientas generadoras de datos.
- **Modelación:** facilitó la creación de abstracciones para lograr un mejor entendimiento del problema a resolver y así obtener un modelado de la aplicación.
- **Analítico-Sintético:** se seleccionó para el análisis del objeto de estudio, de manera que se puedan describir sus características generales y se utilizó con el propósito de analizar toda la teoría de los medios bibliográficos referentes a la investigación, contribuyendo a un mejor desarrollo en el diseño del sistema.

### **Métodos Empíricos:**

- **Observación:** permitió la recogida de información de cada uno de los conceptos asociados a la problemática planteada y se puso en práctica, al concebir de forma consciente la planificación de la investigación, orientada hacia el logro del objetivo determinado.
- **Entrevista:** se consultaron a especialistas y tutores en el trabajo con la base de datos del Xedro-ERP especialmente en los subsistemas “Contabilidad y Costos y Procesos” con el objetivo de obtener información relevante sobre estos subsistemas, para lograr una herramienta que cumpla con las necesidades requeridas.



- **Medición:** se sigue este método con el objetivo de obtener información numérica acerca de una propiedad o cualidad del objeto, donde se comparan magnitudes medibles y conocidas. Se hará uso de este método en los resultados que brinden las distintas validaciones a la que se sometan los resultados de la investigación.

## **Estructura del trabajo**

El presente trabajo está estructurado de la siguiente forma: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos. A continuación se muestra una breve descripción de cada uno de los capítulos.

### **Capítulo 1: Fundamentación teórica**

El capítulo presenta un grupo de conceptos para lograr un mejor entendimiento del trabajo a desarrollar. Se hace una valoración comparativa sobre las principales herramientas de generación de datos. Además, se realiza un estudio de la metodología, herramientas, tecnologías y lenguajes a utilizar en el desarrollo.

### **Capítulo 2: Requisitos y diseño del sistema**

En este capítulo se realiza una descripción general de la solución propuesta, así como los requisitos funcionales y no funcionales que se tendrán en cuenta para la implementación de la misma, se detallan los aspectos relacionados con su diseño y arquitectura. Se especifican los patrones del diseño aplicados y los artefactos derivados de la metodología de desarrollo de software seleccionada.

### **Capítulo 3: Implementación y validación de la solución**

Este capítulo describe la etapa de implementación. Se elaboran y documentan las pruebas realizadas a la solución propuesta para demostrar el correcto funcionamiento de la misma. Por último se analizan los resultados obtenidos tras la aplicación de la herramienta en un entorno real.

## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

### 1.1 Introducción

En el presente capítulo se abordan diferentes conceptos referentes al dominio del problema, los cuales son imprescindibles para comprender el objetivo y el alcance del trabajo. El capítulo comprende un estudio de definiciones fundamentales para desarrollar la investigación, tales como: datos, base de datos, SGBD, carga de datos, generadores de datos y semántica de datos. Se realiza un estudio de diferentes herramientas de generación de datos existentes en la actualidad. Además se muestran las principales características de las herramientas, tecnologías, lenguaje y metodología a utilizar durante el desarrollo de la investigación.

### 1.2 Conceptos asociados a la investigación

#### 1.2.1 Datos

Los datos son símbolos que describen condiciones, hechos, situaciones o valores. Se caracterizan por no contener ninguna información. Un dato puede significar un número, una letra, un signo ortográfico o cualquier símbolo que represente una cantidad, una medida, una palabra o una descripción. La importancia de los datos está en su capacidad de asociarse dentro de un contexto para convertirse en información. Por si mismos los datos no tienen capacidad de comunicar un significado y por tanto no pueden afectar el comportamiento de quien los recibe. Para ser útiles, los datos deben convertirse en información para ofrecer un significado, conocimiento, ideas o conclusiones. (Rioseco, et al., 2014)

Con el paso del tiempo y el avance tecnológico que ha tenido la humanidad la información que se genera es cada vez mayor. Debido a esto, en muy poco tiempo la información almacenada por las grandes empresas y negocios alcanzaron una dimensión considerablemente voluminosa. Con el elevado crecimiento de esta se presentó la problemática de cómo darle un fin útil. A raíz de esto surge el concepto de Base de Datos.

#### 1.2.2 Bases de datos

El Glosario IEEE de Ingeniería del Software define el término base de datos de la siguiente forma: Una colección de datos interrelacionados almacenados conjuntamente en uno o más ficheros de computadora. (Connexions, 2008)

Las bases de datos son un conjunto de datos almacenados entre los que existen relaciones lógicas y han sido diseñadas para satisfacer los requerimientos de información de una empresa u organización.

En una base de datos, además de los datos, también se almacena su descripción. (Álvarez, 2007)

Otro de los conceptos es aportado por el Dr. Christopher J. Date, quien define a las bases de datos como: un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada. (Date, 2001)

A partir de los conceptos antes mencionados se puede llegar a la conclusión de que una base de datos es un conjunto de datos de un mismo contexto, organizados de tal forma que puedan ser accedidos de forma rápida y sencilla.

Debido al desarrollo tecnológico las bases de datos están en formato digital, solucionando los problemas que existían con el almacenamiento de los datos. Existen programas denominados SGBD que permiten almacenar y posteriormente acceder a los datos.

### **1.2.3 Sistemas Gestores de Base de Datos**

Los SGBD permiten crear y mantener una base de datos. El SGBD actúa como interfaz entre los programas de aplicación y el sistema operativo. El objetivo principal es proporcionar un entorno eficiente a la hora de almacenar y recuperar la información de las base de datos. Este software facilita el proceso de definir, construir y manipular bases de datos para diversas aplicaciones. (Cobo)

Un SGBD es una colección de programas que permiten a los usuarios crear y mantener una base de datos. Sistema software de propósito general que facilita los procesos de definición, construcción y manipulación de la base de datos para distintas aplicaciones. (Pérez, 2010)

Una vez vistos algunos de los principales conceptos de Sistema Gestor de Base de Datos y haciendo un resumen de los mismos se puede llegar a la conclusión que los SGBD son un conjunto de aplicaciones que permiten a los usuarios definir, crear y mantener la base de datos y proporciona un acceso controlado a la misma.

Uno de los SGBD usados hoy en día es PostgreSQL, este es distribuido bajo licencia libre y su código fuente se encuentra disponible. Es considerado uno de los SGBD de código abierto más potente del mercado y en sus últimas versiones se encuentra al nivel de los principales gestores propietarios.

### **1.2.4 Sistema Gestor de Base de Datos PostgreSQL**

PostgreSQL dispone de versiones para prácticamente todos los sistemas operativos. Tiene soporte para llaves foráneas, uniones, vistas y disparadores. Incluye la mayoría de los tipos de datos de SQL92

y SQL99 y, así mismo. Tiene interfaces de programación nativas para C/C++, Java, .Net, Perl, PHP, Python, Ruby, Tcl y ODBC, entre otros, y una gran documentación. (Martinez, 2009)

Comenzó como un proyecto denominado Ingres en la Universidad Berkeley de California y fue más tarde desarrollado comercialmente por la Relational Technologies/Ingres Corporation. Posee muchas características, que tradicionalmente sólo se podían ver en productos comerciales de alto nivel, dentro de estas están: (Pérez, 2010)

- El tamaño máximo de las BD es ilimitado.
- Es un excelente optimizador de consultas.
- Permite el uso de particiones para la mejora de la eficiencia de replicación.
- Muchas de sus versiones admiten la administración de las BD distribuidas.
- Dispone de tipos de datos para aplicaciones específicas
- Cuenta con una comunidad de soporte que da mantenimiento al software de forma directa.
- Usa la estrategia de almacenamiento de filas, llamada Control de Concurrencia de Múltiples Versiones (MVCC, por sus siglas en inglés), para conseguir una mejor respuesta ante magnos volúmenes de datos.
- Es multiplataforma y el código fuente es libre.

A pesar de todo lo visto anteriormente, las bases de datos tienen como característica que a medida que va incrementándose el volumen de datos almacenados en las mismas, el tiempo de respuesta a las peticiones realizadas por los usuarios se hace más lento.

A raíz de esto se hace vital la realización de pruebas cuando estas cuentan con un elevado volumen de información, pues como se dijo el comportamiento generalmente no es el mismo. Para hacer estas pruebas viables se necesita elevado volumen de datos que se asemeje a la realidad, crear dicho volumen manualmente acarrea un gasto de tiempo y esfuerzo considerable. Por lo que surge el concepto de generación automática de datos.

### **1.2.5 Generación de datos**

La generación de datos de prueba, es el proceso de crear un conjunto de datos para comprobar la adecuación de las aplicaciones nuevas o revisadas de software. Puede ser los datos reales que se ha tomado de las operaciones anteriores o datos artificiales creados para este propósito. (Korel, 1990)

La generación de datos no es más que la obtención de datos con consistencia pero sin valor real alguno, que permite simular un entorno parecido a la realidad. La generación de datos es de vital importancia en el desarrollo de pruebas a bases de datos, ya que permite poblarlas con un elevado volumen de datos, que de otra forma, llevarían consigo un gasto considerable de tiempo y de recursos.

Una de las técnicas más utilizadas para la generación de datos es la carga de datos, existen dos formas básicas para realizar la carga de datos: mediante el ingreso individual de los datos, o bien la carga masiva de los datos. El ingreso individual consiste en que cada tipo de información tiene un formato específico para realizar su carga, mientras que la carga masiva se basa en la forma práctica de ingresar una gran cantidad de información a través de planillas de cálculo tipo MS Excel, aunque la carga debe realizarse en formato “Delimitado por comas” (CSV) (Carga de Datos, 2014)

En la actualidad generar datos es una práctica muy utilizada por los desarrolladores; y aunque no se ha difundido mucha información sobre el tema, el número de generadores de datos desarrollados sigue creciendo exponencialmente. Para poder realizar esta tarea de forma sencilla y al mismo tiempo de manera eficiente surgen las herramientas generadoras de datos. Estas permiten la generación de grandes volúmenes de datos de forma sencilla y en un corto período de tiempo.

### **1.3 Herramientas de generación de datos**

Contar con una base de datos poblada con un gran volumen de datos, es imprescindible a la hora de probar un software que implique la utilización de bases de datos. Existen diversas herramientas que realizan esta función para el Sistema Gestor de Base de Datos PostgreSQL, a continuación se verán algunas de las más utilizadas mundialmente.

#### **1.3.1 EMS Data Generator for PostgreSQL**

Es una aplicación para generar datos de prueba para varias tablas de bases de datos de PostgreSQL a la vez, la aplicación permite seleccionar tablas y campos para generar datos, establecer intervalos de valores, generar campos de gráfico por máscara, obtener listas de valores de consultas SQL y muchas otras características para generar datos de prueba en una forma simple y directa.

La aplicación tiene como limitación principal que no es de código abierto y tiene una licencia Shareware, por lo que permite evaluar de forma gratuita el producto, pero hay que pagar para usar de forma completa el software. Además no tiene en cuenta las relaciones entre tablas, por lo que en el caso de que una tabla tenga una llave extranjera, no podrá ser llenada. (EMS, 2014)

## 1.3.2 Datanamic Data GeneratorMultiDB

La herramienta permite a los desarrolladores poblar bases de datos fácilmente con grandes volúmenes de datos. Se puede utilizar para generar datos de prueba partiendo de cero o de datos existentes y es posible además generar un script de inserción, en el caso de que se quiera poblar la base de datos en otro momento.

Como su nombre lo indica MultiDB, esta herramienta es capaz de trabajar con múltiples sistemas gestores de bases de datos: Oracle, MySQL, MS SQL Server, PostgreSQL y bases de datos de MS Access. (Datanamic, 2014). Al igual que EMS Data Generator, Datanamic Data GeneratorMultiDB es privativo y solo posee gratis una versión de prueba.

## 1.3.3 Postgen

Fue creado en el Centro de Desarrollo de Software de Santiago de Cuba a partir de la necesidad de una herramienta de código abierto, para generar datos de una base de datos. Este es multiplataforma, de fácil manejo para el usuario. Permite configurar valores por rangos, insertar datos en varias tablas y el control automático sobre la integridad de referencia para la generación ligada de los datos de las tablas.

Esta herramienta tiene entre sus características que solo es funcional para versiones inferiores a la 9.0 del gestor PostgreSQL, fue implementada utilizando el lenguaje de programación C++, específicamente el framework QT. Entre las técnicas de generación de datos que soporta, solo está la aleatoria, siendo esto último una de sus principales limitantes. Además esta herramienta tiene como deficiencia fundamental que solo es capaz de generar datos de tipo texto y numérico. (Riviera, et al.)

## 1.3.4 Comparación entre las herramientas analizadas

La tabla que se muestra a continuación ofrece una comparación entre las herramientas de generación de datos estudiadas, atendiendo a una serie de indicadores o características que presentan estas aplicaciones, agregándole características propias de la aplicación a desarrollar. Estos indicadores son: licencia, relación entre tablas, tipos de datos que soporta, semántica de datos, versiones que soporta, multiplataforma, origen de datos, ayuda y creación de tipos de datos.

Herramientas generadoras de datos ficticios	EMS Data Generator para PostgreSQL	Postgen	Datanamic Data Generator MultiDB
---	------------------------------------	---------	----------------------------------

## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Plataforma	Desktop	Desktop	Web
Ventajas	<p>Ofrece una aplicación de consola, que permite generar datos mediante plantillas de generación.</p> <p>Ofrece una Interfaz de asistente fácil de usar.</p>	<p>Es multiplataforma y de fácil manejo para el usuario.</p>	<p>Puede leer automáticamente la base de datos y detectar cuando una columna o tabla ha sido cambiada o suprimida y luego aplica estos cambios en las tablas del proyecto.</p>
Desventajas	<p>No es de código abierto y utiliza una Licencia shareware.</p> <p>No soporta los datos incluidos en las últimas versiones del gestor.</p> <p>No tiene en cuenta las relaciones entre tablas, por lo que en el caso de que una tabla tenga una llave extranjera, no podrá ser llenada.</p> <p>No tiene en cuenta la semántica de los datos.</p>	<p>Sólo es capaz de generar datos de tipo texto y numérico.</p> <p>No tiene en cuenta la semántica de los datos.</p>	<p>Es una aplicación privativa.</p> <p>No tiene en cuenta la semántica de los datos.</p>
Tipos de generación.	<p>1- De una lista.</p> <p>2- Incremental</p> <p>3- Aleatoria</p>	<p>1- Aleatoria</p>	<p>1- Aleatoria</p> <p>2- De una lista</p> <p>3- De un archivo externo.</p>

			4- De otra tabla
--	--	--	------------------

Tabla 1 Comparación entre las herramientas analizadas.

Luego de un análisis de las herramientas generadoras de datos existentes en la actualidad se concluye que se desarrollará una herramienta generadora de datos, pues la mayoría de estas son privativas y no cumplen con las restricciones del sistema como; tener en cuenta la semántica de datos, las dependencias entre las tablas y crear nuevos tipos de datos, requisitos fundamentales para la aplicación. Por otra parte el Postgen aunque no es propietaria su principal deficiencia es que genera dos tipos de datos, numéricos y textos además de que no cumple con las características anteriormente expuestas. Con el objetivo de lograr una aplicación más completa, se tendrán en cuenta algunas funcionalidades y características de las herramientas vistas anteriormente, como son el diseño de las interfaces y las técnicas de generación de datos, de estas últimas se hizo un compendio de las técnicas que utilizan estas herramientas y las mismas serán implementadas en la aplicación como son: de una lista, aleatoria, de un archivo externo y de otra tabla, además se le incorporará la técnica mediante un formato logrando de esta forma una herramienta más completa y brindarle más opciones a los usuarios.

## 1.4 Metodología de desarrollo

El proceso de desarrollo de software no es una tarea fácil, se debe contar con un proceso bien detallado y para esto se necesita aplicar una metodología que sea capaz de llevar a cabo el control total del producto. Las metodologías de desarrollo de software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. (Letelier, et al., 2006)

Las metodologías se dividen en dos grupos, tradicionales (pesadas) y ágiles (ligeras). Las tradicionales, se centran en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, pretendiendo prever todo de antemano además dependen de un equipo de desarrollo bastante grande. En las ágiles es más importante lograr que un producto de software se realice con la calidad requerida que hacer una buena documentación, en este tipo de metodología el cliente está presente en todo momento y colabora con el proyecto que posee un equipo de desarrollo pequeño como un miembro más. (Letelier, et al., 2006)



La metodología de desarrollo de software seleccionada por el equipo de trabajo incurre en un enfoque ágil; Programación Extrema (XP), se preocupa más en el desarrollo exitoso del producto que en generar una documentación detallada del mismo, por lo cual es capaz de adaptarse a los cambios de requisitos en cualquier punto del ciclo de vida del proyecto.

### 1.4.1 Extreme Programming (XP)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Además se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (Joskowicz, 2008)

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre Kent Beck, el padre de XP, describe la filosofía de XP sin cubrir los detalles técnicos y la implantación de las prácticas. (Joskowicz, 2008) Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación presentaremos algunas ventajas, desventajas y fases esenciales de XP:

#### **Ventajas:**

- Programación organizada.
- Menor tasa de errores.
- Satisfacción del programador.

#### **Desventajas:**

- Es recomendable emplearlo solo en proyectos a corto plazo.
- Altas comisiones en caso de fallar.

#### **Fases de la Metodología XP.**

El ciclo de vida ideal de XP propuesto por Kent Beck consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

#### **Fase I: Exploración.**

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología. (Letelier, et al., 2006)

### **Fase II: Planificación de la Entrega.**

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la “velocidad” de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. (Letelier, et al., 2006)

### **Fase III: Iteraciones.**

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fueren la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. (Letelier, et al., 2006)

### **Fase IV: Producción.**

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento). (Letelier, et al., 2006)

### **Fase V: Mantenimiento**

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura. (Letelier, et al., 2006)

### **Fase VI: Muerte del Proyecto.**

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo. (Letelier, et al., 2006)

Se decide utilizar XP, ya que se adecúa al desarrollo del siguiente trabajo y a los contextos en que se realiza el mismo. El cliente forma parte del equipo de desarrollo de la aplicación, logrando una retroalimentación continua y corrección de errores. El sistema a desarrollar es pequeño por lo que todo el trabajo se realiza por una pareja de programadores. En la medida que progresa el proyecto, el cliente puede agregar nuevas historias de usuario, modificarlas, o eliminarlas. Permite al equipo la modificación de sus planes en correspondencia con los anteriores. El proyecto está ajustado para ser desarrollado en el menor tiempo posible. Además permite probar los resultados que se vayan obteniendo en cada iteración.

### **1.5 Herramientas, lenguajes y tecnologías utilizadas**

En todo proceso investigativo es necesario la utilización de sistemas de soporte que permitan organizar, facilitar, agilizar y automatizar las tareas generadas durante el transcurso de la investigación. Las herramientas, lenguajes y tecnologías empleadas que se describen a continuación son de vital importancia para una correcta realización de la misma.

## 1.5.1 Lenguaje de modelado

El lenguaje unificado de modelado (UML) es un lenguaje estándar de modelado para software, un lenguaje para la visualización, especificación, construcción y documentación de los artefactos de sistemas en los que el software juega un papel importante. Básicamente UML permite a los desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados. (OMG, 2014)

UML posee grandes propiedades como son: (OMG, 2014)

- Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.
- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.

## 1.5.2 Herramienta CASE

Las herramientas CASE, del español Ingeniería de Software Asistida por Computadoras son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. (González, 2012)

Las herramientas CASE poseen grandes ventajas que permiten aumentar la productividad en el desarrollo de un software: (González, 2012)

- Verificar el uso de todos los elementos en el sistema diseñado.
- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos.

## 1.5.2.1 Visual Paradigm

Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad. Permite construir diagramas de clases y realizar el proceso de ingeniería inversa. (Visual Paradigm, 2014)

El Visual Paradigm ofrece diversas características como son: (Visual Paradigm, 2014)

- Entorno de creación de diagramas para UML 2.1
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.

Se selecciona Visual Paradigm como herramienta para el modelado UML pues tiene entre sus características que permite trabajar de forma colaborativa, hacer un trabajo organizado y ágil. Posibilita la realización de los diagramas necesarios para el desarrollo y mejor entendimiento de la aplicación. Además permite realizar ingeniería inversa a partir del código fuente, lo que facilita el proceso de desarrollo de la herramienta y agiliza el trabajo a los desarrolladores.

## 1.5.3 Lenguaje de programación

Los lenguajes de programación son un conjunto de símbolos junto a un conjunto de reglas para combinar dichos símbolos que se usan para expresar programas. Constan de un léxico, una sintaxis y una semántica. (Programación, 2009)

Partiendo de las características de la aplicación, se hace necesaria la selección de un lenguaje mediante el cual se pueda cumplir con los requisitos propuestos. Actualmente existen muchos lenguajes para el desarrollo de aplicaciones surgidos a partir de las tendencias y necesidades de los escenarios. El análisis se centró fundamentalmente el lenguaje Java.

## 1.5.4 Lenguaje Java

Es un lenguaje de programación sencillo, orientado a objetos, de propósito general e independiente de la plataforma de desarrollo. Este lenguaje posibilita la integración externa, como el uso de herramientas,

métodos y funcionalidades desarrolladas por otros programadores. Esto supone una ventaja tanto en el ámbito del desarrollo como en la repercusión final de un proyecto. (JavaHispano, 2014)

**Entre las características principales de Java se encuentran:** (JavaHispano, 2014)

- **Es robusto:** realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos ayuda a detectar errores lo antes posible, en el ciclo de desarrollo.
- **Es de arquitectura neutral:** para establecer Java como parte integral de la red, el compilador compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará.
- **Es dinámico:** se beneficia todo lo posible de la tecnología orientada a objetos. No intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución.
- **Es seguro:** el sistema de Java tiene ciertas políticas que evitan que se puedan codificar virus con este lenguaje.
- **Es multihilo:** un lenguaje que soporta múltiples hilos, es decir puede ejecutar diferentes líneas de código al mismo tiempo.

Se decide utilizar Java como lenguaje de programación ya que es uno de los lenguajes de programación más potentes en el trabajo con la programación multihilo, una de las funcionalidades que más se utiliza en el desarrollo de aplicaciones para la generación de datos. Además el equipo de desarrollo cuenta con experiencia suficiente en el desarrollo de aplicaciones en este lenguaje. Otra de las razones es que cuenta con el driver JDBC-PostgreSQL para el trabajo con base de datos PostgreSQL, el cual permite automatizar una gran parte del trabajo con este gestor.

### 1.5.5 Entorno de desarrollo integrado

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés), es una herramienta que permite a los desarrolladores de software escribir sus programas en uno o más lenguajes. Consiste básicamente en una plataforma en la que se integran un editor de código, un compilador, un depurador y una interfaz gráfica de usuario. ( Entornos de programación, 2012)

#### 1.5.5.1 NetBeans

NetBeans es una aplicación de código abierto diseñada para el desarrollo de aplicaciones portable entre las distintas plataformas, haciendo uso de la tecnología Java. Dispone de soporte para crear

interfaces gráficas de forma visual, desarrollo de aplicaciones Web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles y funcionalidades ampliables mediante la instalación de paquetes. (Netbeans, 2014)

**Soporta varios lenguajes, entre ellos:** (Netbeans, 2014)

- Java
- C/C++
- Ruby on Rails
- PHP

**Características de NetBeans:** (Netbeans, 2014)

- Instalación y actualización simple.
- Diseñador visual de formularios para Swing GUI.
- Características visuales para el desarrollo de aplicaciones Desktop.
- Elevado completamiento de código.

Se decide utilizar NetBeans como entorno de desarrollo debido a que este soporta el lenguaje de programación java, una de sus particularidades es que permite un desarrollo rápido de las interfaces de usuarios, lo que agiliza el proceso de implementación de la herramienta. Además de que cuenta con un elevado completamiento de código y una ayuda para cada uno de los métodos propios del lenguaje (Javadoc).

### 1.6 Patrones de diseño

Un patrón de diseño consiste en la descripción de un problema y su solución, el cual recibe un nombre y puede emplearse en otros contextos. Indica o puede tener una sugerencia de cómo aplicarlo y utilizarlos en otros contextos y en situaciones nuevas. (Larman, 1999)

Dentro de estos patrones se encuentran los GRASP (Patrones de Software para la asignación General de Responsabilidad) los que constituyen un apoyo para la enseñanza, pues ayudan a entender el diseño de objetos. Estos patrones permiten elegir las clases adecuadas y decidir cómo debe interactuar, ayuda a asignar las responsabilidades de cada clase. (Larman, 1999)

También se encuentran los patrones GOF los cuales se encargan de describir soluciones simples y de una manera elegante a problemas en un contexto particular, dentro del diseño de software orientado a objetos. Los patrones GOF recopilan una serie de patrones de diseños, agrupados en tres categorías: de creación, de estructura y de comportamiento. (Larman, 1999)

En el Capítulo 2 se muestra una explicación de los patrones de diseños utilizados en la implementación del sistema.

## 1.7 Métricas utilizadas

### Métrica para validar el diseño

Una métrica es un instrumento que cuantifica un criterio y persigue comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado al nivel del proyecto. Para la evaluación de la calidad del diseño propuesto para la herramienta generadora de datos se hizo un estudio de las métricas básicas inspiradas en la calidad del diseño orientado a objeto, en el mismo se incluyen los atributos de calidad que permiten medir el diseño propuesto. (Vázquez, et al., 2004)

Las métricas concebidas como instrumento para evaluar la calidad del diseño y su relación con los atributos de calidad definidas son las siguientes:

**Tamaño Operacional de Clase (TOC):** Esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación. Por otro lado, en cuanto menor sea el valor medio para el tamaño, más probable es que las clases tengan menos responsabilidad y complejidad y más nivel de reutilización. (Lorenz, et al., 1994)

	Categoría	Umbrales
Responsabilidad	Baja	$\leq$ Prom.
	Media	Entre Prom. y $2^*$ Prom.
	Alta	$> 2^*$ Prom.



## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Complejidad implementación	Baja	$\leq$ Prom.
	Media	Entre Prom. y $2^*$ Prom.
	Alta	$> 2^*$ Prom.

Reutilización	Baja	$> 2^*$ Prom.
	Media	Entre Prom. y $2^*$ Prom.
	Alta	$\leq$ Prom.

Tabla 2 Umbrales de la métrica Tamaño Operacional de Clase.

**Relaciones entre Clases (RC):** Esta métrica se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases, y es inversamente proporcional al grado de reutilización de las mismas. (Lorenz, et al., 1994)

	Categoría	Umbrales
Acoplamiento	Ninguna	0
	Bajo	1
	Medio	2
	Alto	$>2$

Complejidad Mantenimiento	Baja	$\leq$ Prom.
	Media	Entre Prom. y $2^*$ Prom.

	Alta	$> 2 * Prom.$
Reutilización	Baja	$> 2 * Prom.$
	Media	Entre Prom. y $2 * Prom.$
	Alta	$\leq Prom.$
Cantidad de Prueba	Baja	$\leq Prom.$
	Media	Entre Prom. y $2 * Prom.$
	Alta	$> 2 * Prom.$

Tabla 3 Umbrales de la métrica Relaciones entre clases

## 1.8 Pruebas de calidad

Las pruebas de software son una actividad en la cual el sistema es ejecutado bajo condiciones específicas, para demostrar que tiene o no, la madurez necesaria para ser implantado. Permiten determinar la calidad del producto, detectar todo posible mal funcionamiento y comprobar el grado de cumplimiento de las especificaciones iniciales del sistema. Se considera una prueba exitosa, si se demuestran deficiencias en el software. La prueba no puede asegurar la ausencia de defectos; solo puede demostrar que existen defectos en el software (Pressman, 2006).

### 1.8.1 Justificación de las pruebas a utilizar

La metodología XP propone para validar las necesidades de los usuarios y dirigir la implementación las pruebas unitarias y las de aceptación. Las unitarias están encomendadas a verificar el código, son diseñadas por los programadores, garantizan que un determinado módulo cumpla con un comportamiento esperado antes de ser integrado al sistema. Se emplean cuando la implementación es complicada y la interfaz de un método no es clara. Las pruebas de aceptación están destinadas a evaluar la funcionalidad requerida del software. Son definidas y diseñadas por el cliente, permitiendo

validar todas las funcionalidades. Posibilitan que los programadores estén al tanto de lo que resta por realizar. Su principal objetivo es asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. (Joskowicz, 2008)

Para verificar que el sistema desarrollado cumple con las funciones específicas para las que se ha creado se deciden utilizar los dos tipos de pruebas que define la metodología XP, pruebas unitarias y pruebas de aceptación.

### ¿Por qué usar pruebas unitarias?

Las pruebas unitarias se realizan para controlar el funcionamiento de pequeñas porciones de código. Generalmente son realizadas por el mismo programador, debido que al conocer con mayor detalle el código, se les simplifica la tarea de elaborar conjuntos de datos de prueba para testearlo.

En este tipo de prueba existen tres enfoques de diseños de casos de pruebas, los cuales permiten ir probando las distintas funcionalidades de forma independiente y de esta forma eliminar futuros errores:

- **Enfoque estructural o de caja blanca:** esta consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba.
- **Enfoque funcional o de caja negra:** consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos.
- **Enfoque aleatorio:** consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba. (Joskowicz, 2008)

### ¿Por qué usar pruebas de aceptación?

Como anteriormente se había explicado, la metodología XP se enfoca en satisfacer las necesidades del cliente, es un tipo de prueba que cumple con lo planteado. Además estas pruebas las hace el cliente para verificar si su producto está al 100% correcto. Se utilizará el enfoque de prueba funcional o de caja negra y empleando la técnica de partición equivalente, ya que mediante estas pruebas se obtiene un producto final con gran calidad y principalmente se logra cumplir con las especificaciones tratadas con el cliente.

### 1.9 Conclusiones parciales

Luego del desarrollo del presente capítulo se puede arribar a las conclusiones siguientes:

## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

---

- Se analizaron los principales conceptos asociados al tema tales como base de datos, gestor de base de datos y generación de datos para un mejor entendimiento del dominio del problema.
- Se hizo un estudio de algunas de las herramientas existentes en el mundo para la generación de datos, EMS Data Generator para PostgreSQL, Postgen y Datanamic Data Generator MultiDB con el objetivo de identificar las técnicas de generación de datos a implementar.
- Se identificaron las principales herramientas y tecnologías a usar para el desarrollo del Generador de Datos, donde se eligió UML 2.0 como lenguaje de modelado, Visual Paradigm 8.0 como herramienta CASE, Java como lenguaje de programación y NetBeans7.3.1 como entorno de desarrollo.
- Para la comprobación de la calidad del sistema se describieron los dos métodos de prueba que define la metodología XP: pruebas unitarias y pruebas de aceptación.
- Para un mejor entendimiento y organización de las clases del diseño fueron descritos los patrones GRASP y GOF, los que serán utilizados durante el diseño de la aplicación.

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

### 2.1 Introducción

En el presente capítulo se hace una descripción del desarrollo de la herramienta, detallando el análisis, diseño y la arquitectura, así como los requisitos funcionales y no funcionales que se tendrán en cuenta para la implementación de la solución propuesta. Además se muestran explícitamente las historias de usuarios y los requisitos no funcionales, también serán generados todos los artefactos propuestos por la metodología XP correspondientes a la etapa de diseño y los diferentes patrones de arquitectura y de diseño que serán utilizados.

### 2.2 Propuesta del sistema

Para darle respuesta al problema a resolver expuesto en el capítulo anterior, se propone el desarrollo de una aplicación de escritorio que genere datos para los subsistemas de “Contabilidad y Costos y Procesos”. Además la aplicación tiene como una de sus ventajas que podrá ser utilizada por cualquier sistema desarrollado sobre la base de datos PostgreSQL.

El usuario creará un proyecto el cual se conectará a la base de datos del Xedro-ERP, luego podrá seleccionar el esquema con el cual va a trabajar. Después seleccionará una tabla y le generará el árbol de dependencias. El sistema le posibilitará al usuario seleccionar el origen de los datos (mediante una lista de valores, un archivo en formato csv o txt, una consulta SQL a una base de datos local o remota, aleatorio según el tipo de dato y mediante formato definido por el usuario). El sistema permitirá guardar un scripts SQL con los valores generados. Además podrá guardarse toda la configuración hecha en el sistema y en caso de que la base de datos fuera modificada en algún momento la aplicación permitirá actualizar la estructura del proyecto creado sin perder la información guardada.

### 2.3 Historias de usuarios

La historia de usuario (HU) es la técnica utilizada en XP para especificar los requisitos del software; se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. (Letelier, et al., 2006)

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

Luego de obtener las principales funcionalidades del sistema se identificaron 26 historias de usuario. A continuación se muestra una breve descripción de dos historias de usuario arquitectónicamente significativas y el resto se encuentran especificadas en el anexo. [\(Ver anexo\)](#)

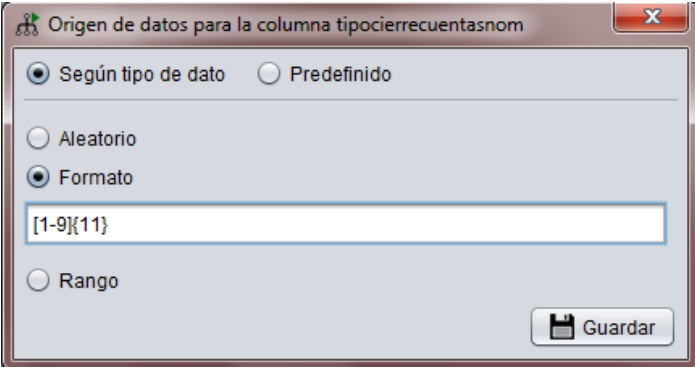
Historia de Usuario	
<b>Número:</b> 10	<b>Nombre Historia de Usuario:</b> Generar datos mediante formato.
<b>Modificación de Historia de Usuario Número:</b> Ninguna	
<b>Usuario:</b> Reinier González Medinilla	<b>Iteración Asignada:</b> 1
<b>Prioridad en Negocio:</b> Muy Alta	<b>Puntos Estimados:</b> 2
<b>Riesgo en Desarrollo:</b> Muy Alta	<b>Puntos Reales:</b> 2
<b>Descripción:</b> Se adicionan los datos de una columna a las bases de datos, una vez que el usuario seleccione la técnica de generación que desee utilizar y configure los parámetros definidos por esta.	
<b>Observaciones:</b>	
<b>Prototipo de interfaz:</b>	

Tabla 4 Historia de usuario Generar datos mediante un formato.

Historia de Usuario	
<b>Número:</b> 8	<b>Nombre Historia de Usuario:</b> Generar datos mediante una lista.

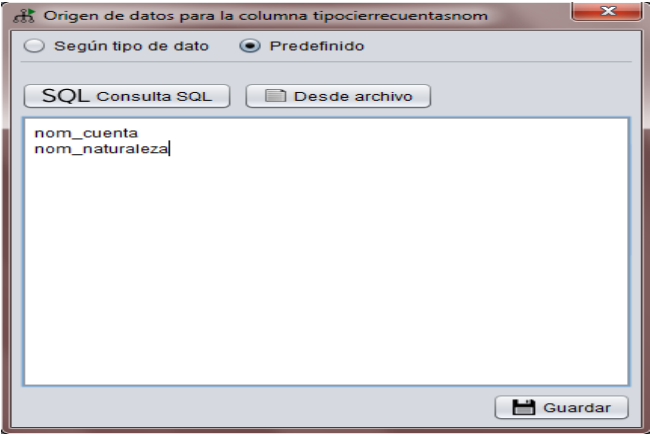
<b>Modificación de Historia de Usuario Número:</b> Ninguna	
<b>Usuario:</b> Reinier González Medinilla	<b>Iteración Asignada:</b> 1
<b>Prioridad en Negocio:</b> Muy Alta	<b>Puntos Estimados:</b> 2
<b>Riesgo en Desarrollo:</b> Muy Alta	<b>Puntos Reales:</b> 2
<b>Descripción:</b> El usuario podrá definir los elementos que desea generar a través de una lista de valores.	
<b>Observaciones:</b>	
	
<b>Prototipo de interfaz:</b>	

Tabla 5 Historia de usuario generar datos mediante una lista.

### 2.3.1 Lista de reserva de productos

La lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir la aplicación que se desea realizar, ordenados según la prioridad de implementación, ubicados en Muy Alta, Alta, Media y Baja, en esta última aparecen los requisitos de menor complejidad, además de los requisitos no funcionales del sistema a desarrollar. Indica la estimación de cada uno de ellos, su implementación por semanas y quien realizó la estimación. (Letelier, et al., 2006)

A continuación se muestra la lista de reserva de productos

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

	Descripción	Estimación	Estimado por
<b>Prioridad: Muy Alta</b>			
1	Crear árbol de dependencia.	1.5	Analista
2	Eliminar tabla del árbol de dependencia.	0.5	Analista
3	Agregar tabla al árbol de dependencia	0.5	Analista
4	Visualizar árbol de dependencia	1.5	Analista
5	Actualizar árbol de dependencia	0.5	Analista
6	Abrir árbol de dependencia	0.5	Analista
7	Guardar árbol de dependencia	0,5	Analista
8	Generar datos mediante una lista	2	Analista
9	Generar datos aleatorios según tipo de datos	2	Analista
10	Generar datos mediante formato	2	Analista
11	Generar datos mediante una consulta	2	Analista
12	Abrir consulta desde un fichero	1	Analista
13	Guardar consulta en un fichero	1	Analista
14	Visualizar resultados de la consulta	1	Analista
15	Generar datos a través de un archivo	2	Analista
<b>Prioridad : Alta</b>			
16	Crear proyecto	1	Analista



## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

17	Guardar proyecto	1	Analista
18	Abrir Proyecto	0.5	Analista
19	Listar proyectos y arboles dependencias recientes.	0.5	Analista
20	Visualizar árbol de objetos	1.5	Analista
21	Actualizar árbol de objetos	0.5	Analista
22	Buscar elemento en árbol de objetos	1	Analista
23	Crear scripts de datos generados	1.5	Analista
<b>Prioridad: Media</b>			
24	Crear conexión	1	Analista
25	Modificar conexión	0.5	Analista
26	Modificar configuración del proyecto	0.5	Analista
<b>Requisitos no funcionales</b>			
1	<p><b>Software:</b> La herramienta será multiplataforma, lo que le permite ser ejecutada sobre varios sistemas operativos.</p> <p>En las máquinas clientes para poder ejecutar la aplicación es necesario tener instalada la máquina virtual de java JVM.</p>		Desarrollador
2	<p><b>Hardware</b></p> <p><b>Mínimo:</b> La estación de trabajo debe contar con al menos 1 GB de RAM, un microprocesador con una velocidad superior a 2.0 GHz, así como un espacio libre en el disco duro de 30 MB.</p>		Desarrollador

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

	<b>Recomendado:</b> La estación de trabajo debe contar con al menos 4 GB de RAM, un microprocesador Core i3 con una velocidad superior a 3.1 GHz, así como un espacio libre en el disco duro de 30 MB.	
3	<b>Portabilidad:</b> El sistema será multiplataforma, ejecutándose de manera óptima sin importar el sistema operativo que utilice el usuario.	Analista
4	<b>Usabilidad:</b> Los enlaces y botones serán fáciles de asociar con las operaciones que realizan.	Analista
5	<b>Eficiencia:</b> El sistema ejecutado sobre las prestaciones mínimas del hardware no se demorará más de 3 minutos en cargar y mostrar toda la información referente a los 25 esquemas que conforman la BD del Xedro-ERP.	Desarrollador

Tabla 6 Lista de reservas de productos.

### 2.3.2 Plan de Iteraciones

Una vez finalizadas las historias de usuarios se debe crear un plan de iteraciones, indicando cuáles se desarrollarán en cada iteración del programa. A continuación se muestra el plan de iteraciones para la herramienta de generación de datos, la cual quedó definida que se iba a desarrollar en un total de 2 iteraciones.

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración Total
<b>Iteración 1</b>	En esta iteración se implementarán las HU 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. Estas historias son de prioridad muy alta pues son las que mayor incidencia tienen en la funcionalidad de la aplicación. Por tal motivo son desarrolladas en esta iteración.	HU #1, HU #2, HU #3, HU #4, HU #5, HU #6, HU #7, HU #8, HU #9, HU #10, HU #11, HU #12, HU #13, HU #14, HU #15.	18.5 semanas

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

<b>Iteración 2</b>	En la 2da iteración se desarrollarán las HU 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26 las cuales son de menor complejidad para el desarrollo. Estas tienen prioridad Alta y Media respectivamente.	HU #16, HU #17, HU #18, HU #19, HU #20, HU #21, HU #22, HU #23, HU #24, HU #25, HU #26.	8.5 semanas
--------------------	---	---	-------------

Tabla 7 Plan de iteración.

### 2.3.3 Tareas de ingeniería

Al inicio de la primera iteración se debe tener claro qué es lo que se debe codificar. Todo el trabajo de la iteración es expresado en tareas de programación, las cuales se realizan para especificar las acciones llevadas a cabo por los programadores en cada historia de usuario, aunque estas describen a un alto nivel lo que se necesita implementar, no brindan los detalles suficientes para poder hacer una codificación correcta.

Según el Plan de iteraciones las historias de usuario se agruparon en dos iteraciones. A continuación se muestra un ejemplo de dos tareas de ingeniería que responden a dos historias de usuarios arquitectónicamente significativas, ([Ver Anexo](#)) para las restantes tareas de programación.

Tarea de Ingeniería	
<b>Número Tarea:</b> 8	<b>Numero Historia de Usuario:</b> HU #8
<b>Nombre Tarea:</b> Generar datos mediante formato	
<b>Tipo Tarea:</b> Desarrollo	<b>Puntos Estimados:</b> 2
<b>Fecha Inicio:</b> 15/02/2014	<b>Fecha Fin:</b> 1/3/2014
<b>Programador Responsable:</b> José Luis Bizet Romero	

**Descripción:** Esta función obtendrá por parámetros formato creado por el usuario. Luego genera los datos en dependencia del formato antes creado.

Tabla 8 Tarea de ingeniería Generar datos mediante un formato.

Tarea de Ingeniería	
<b>Número Tarea:</b> 2	<b>Numero Historia de Usuario:</b> HU # 7
<b>Nombre Tarea:</b> Generar datos mediante una lista	
<b>Tipo Tarea:</b> Desarrollo	<b>Puntos Estimados:</b> 2
<b>Fecha Inicio:</b> 2/03/2014	<b>Fecha Fin:</b> 15/3/2014
<b>Programador Responsable:</b> José Luis Bizet Romero	
<b>Descripción:</b> Esta función obtendrá por parámetros los elementos que se desean generar. Estos elementos fueron definidos anteriormente por el usuario en una lista.	

Tabla 9 Tarea de ingeniería Generar datos mediante una lista

### 2.4 Arquitectura de software

La arquitectura de software es la definición y estructuración de una solución que cumple con los requisitos técnicos y operativos. Optimiza atributos que implican una serie de decisiones, tales como la seguridad, el rendimiento y la capacidad de administración. Estas decisiones en última instancia, afectan la calidad de la aplicación, el mantenimiento, el rendimiento y el éxito global. (Pressman, 2006)

#### 2.4.1 Estilos arquitectónicos

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software. Estos permiten expresar un esquema de organización estructural esencial para un sistema de software. (Pressman, 2006)

Un estilo expresa un esquema de organización estructural esencial para un sistema de software. En este trabajo se hace uso del estilo arquitectónico en capas logrando que el sistema quede organizado para de esta forma tener un orden lógico en la programación del mismo

### 2.4.2 Arquitectura en capas

La arquitectura en capas permite distribuir el trabajo de una aplicación por niveles. Este basa su trabajo en realizar una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades y funcionalidades que implementan. (Pelaez, 2009)

A continuación se pone un ejemplo de cómo se evidencia el estilo en capas dentro de la aplicación y se hace una breve explicación de cada una de las capas del sistema.

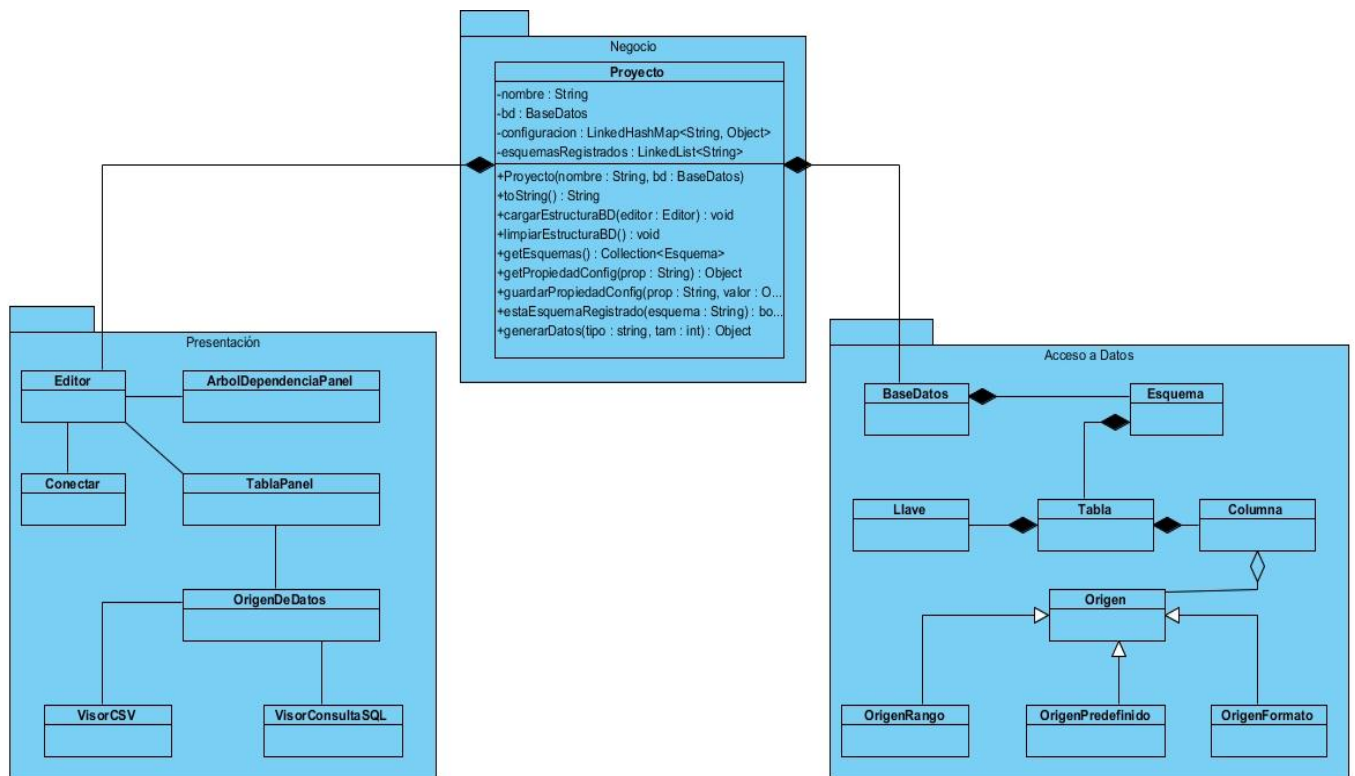


Figura 1 Arquitectura en capas.

**Capa de presentación:** esta capa contiene todos los formularios, en la misma se encuentran las clases encargadas de interactuar con el usuario, esta capa tiene como tarea fundamental los eventos, captura

y visualización de los datos de interés para el usuario. La capa de presentación se comunica únicamente con la capa de negocio. En esta capa se encuentran las clases Editor, Conectar, ArbolDependenciasPanel, TablaPanel, OrigenDeDatos, VisorCSV, VisorConsultaSQL.

**Capa de negocio:** en esta capa se encuentra la lógica del negocio, es el puente entre la primera capa y la tercera, aquí es donde se reciben las peticiones del usuario y se envían las respuestas. En esta capa se establecen todas las reglas que deben cumplirse, se encuentra la clase Proyecto, la cual funge como clase controladora y es la encargada de manejar todo el flujo de negocio de la aplicación.

**Capa de acceso a datos:** en esta capa se encuentran todas las clases que poseen información necesaria, pero que no tienen sentido que existan de forma independiente. Estas clases contienen toda la información que la capa de negocio necesita para realizar las operaciones, cuenta con las clases encargadas de permitir el acceso a la información almacenada dentro de la Base de Datos. En esta capa se encuentran las clases BaseDatos, Esquema, Tabla, Llave, Columna, Origen, OrigenRango, OrigenPredefinido, OrigenFormato.

### 2.5 Patrones de diseño

Los patrones de diseño se han convertido en un elemento básico del diseño orientado a objetos y la programación, los mismos proporcionan soluciones elegantes, fáciles de reutilizar y mantenibles.

#### 2.5.1 Patrones GRASP

GRASP es un acrónimo de General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades). El nombre se eligió para sugerir la importancia de aprender estos principios para diseñar con éxito el software orientado a objetos. Los patrones GRASP son utilizados para describir los principios fundamentales del diseño y la asignación de responsabilidades. (Larman, 1999)

Dentro de los patrones de software GRASP utilizados en la solución figuran los siguientes: Experto, Creador, Controlador, Bajo acoplamiento y Alta cohesión.

A continuación se explica donde se evidencian cada uno de ellos.

**Experto:** El patrón Experto utilizado en el diseño de la aplicación, define como asignar de forma adecuada las responsabilidades en un modelo de clases. Indica que la responsabilidad de la creación de un objeto o la implementación de un método debe recaer en la clase que conoce toda la información necesaria para crearlo, dicho patrón se evidencia en la aplicación en la clase Tabla la cual es la

encargada de conocer toda la información relacionada con las tablas, la misma es la experta en manejar todas las funcionalidades sobre las tablas.

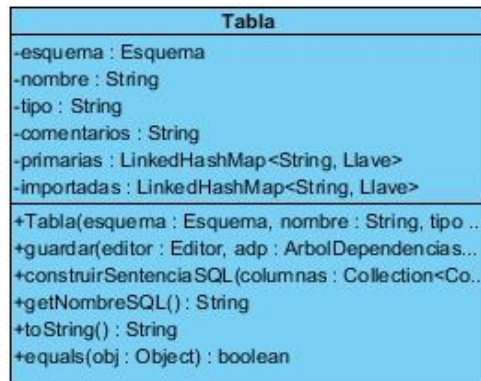


Figura 2 Fragmento del diagrama de clases donde se evidencia el patrón experto.

**Alta cohesión:** Este patrón plantea que la información que almacena una clase debe de ser coherente y debe estar relacionada con la misma. En la aplicación se pone de manifiesto en las clases pertenecientes a la capa de acceso a datos, donde cada una de las clases de esta capa (Base de Datos, Columna, Tabla, Esquema) realiza solo las funcionalidades que le corresponden, lo que permite no sobrecargar las clases con tareas innecesarias.

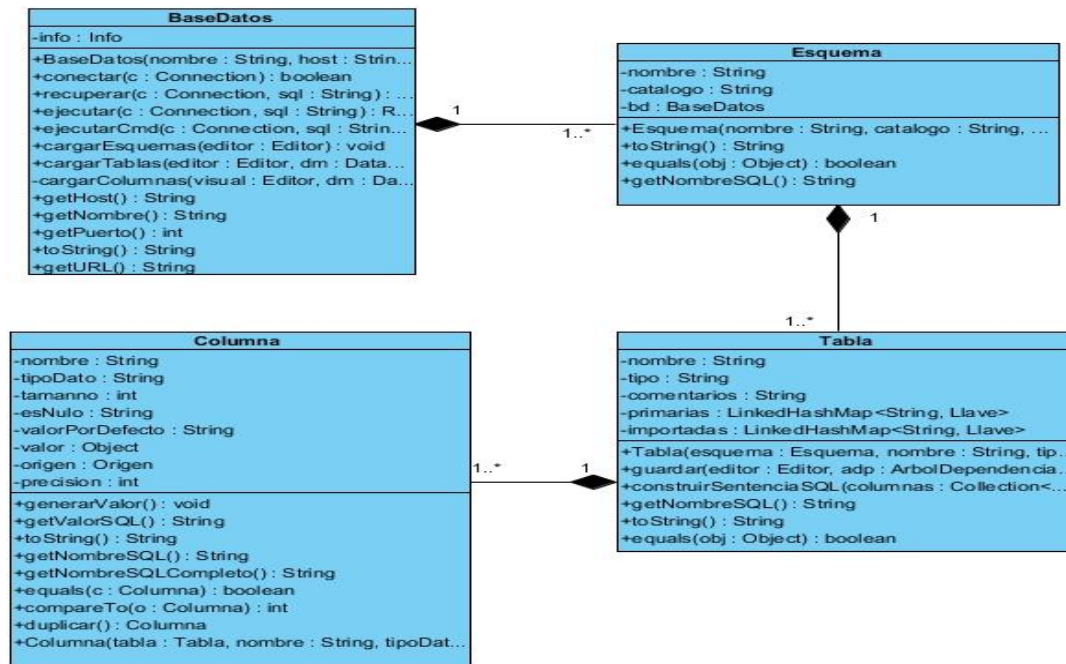


Figura 3 Fragmento del diagrama de clases donde se evidencia el patrón alta cohesión.

**Bajo Acoplamiento:** Cada clase se relaciona lo menos posible con el resto, tratando de que sea flexible a cambios, garantizando que en caso de que se realicen modificaciones en alguna de ellas las afectaciones a las demás sean mínimas, se evidencia en la aplicación entre la clase Proyecto y la clase Base de Datos. La clase controladora Proyecto solo se relaciona con Base de Datos y esta a su vez es la encargada de relacionarse con el resto de las clases de la capa acceso a datos y es la encargada de brindarle los datos a la clase controladora.

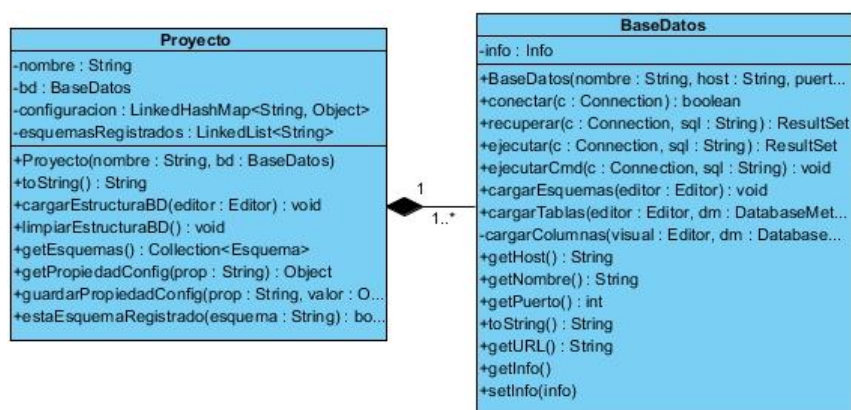


Figura 4 Fragmento del diagrama de clases donde se evidencia el patrón bajo acoplamiento.



**Creador:** 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. El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. La intención básica del patrón es encontrar un creador que necesite conectarse al objeto creado en alguna situación. En la herramienta se evidencia cuando en la clase Base Datos se crea una instancia de la clase Esquema para acceder a las funcionalidades que esta brinda.

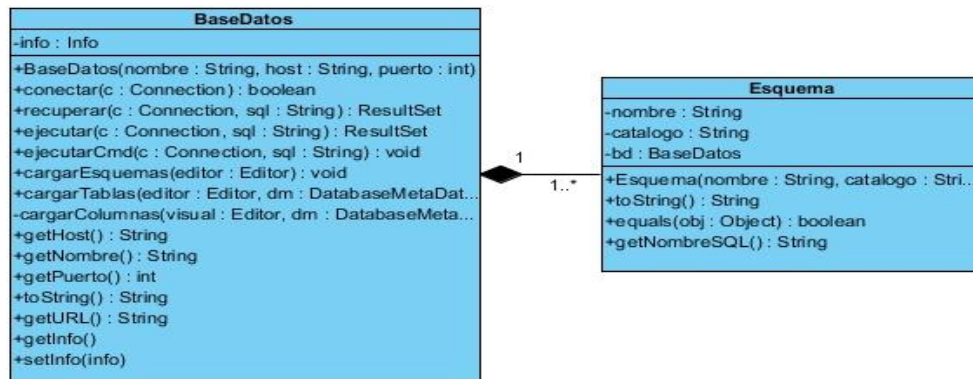


Figura 5 Fragmento del diagrama de clases donde se evidencia el patrón creador.

**Controlador:** Este patrón es el encargado de controlar un evento del sistema, es aquel que sirve para intermediar entre la interfaz y el algoritmo. Dicho patrón se utiliza en la clase Proyecto la cual es la encargada de manejar todo el negocio de la aplicación. En la misma se encuentran todos los métodos para las diferentes formas de generar los datos.

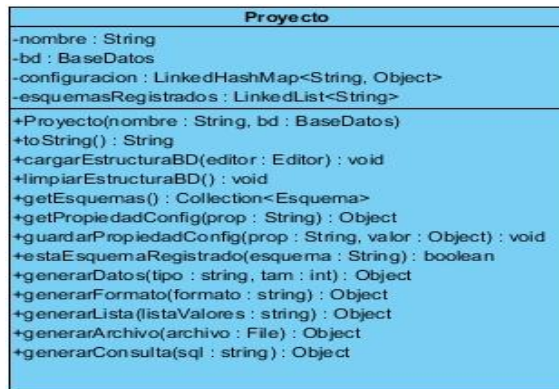


Figura 6 Fragmento del diagrama de clases donde se evidencia el patrón controlador.

### 2.5.2 Patrones GoF

Es la abreviación del grupo Gang of Four, compuesto por Erich Gamma, Richard Helm, Ralph Jhonson y John Vlisodes, quienes en su publicación “Design Patterns” (década de los 90s), describen patrones de diseño comúnmente utilizados y de gran aplicabilidad en problemas de diseño usando el modelado UML. Estos patrones se agrupan en las siguientes categorías: creacionales, estructurales y de comportamiento.

**Comportamiento:** Contribuyen a definir la comunicación e iteración entre los objetos de un sistema. (Software, 2010)

- **Observer (Observador):** Define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos. El uso de este patrón se pone de manifiesto en la clase Tabla y Columna donde al ocurrir un cambio en un objeto de la clase Columna este se actualiza en la clase Tabla.

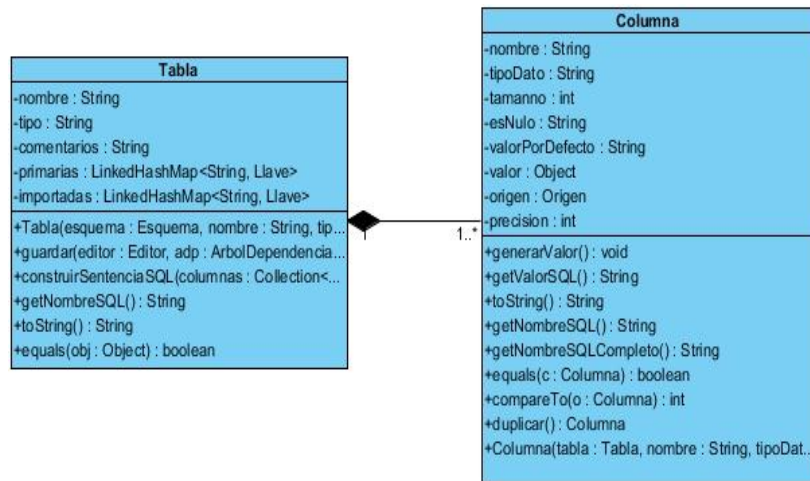


Figura 7 Fragmento del diagrama de clases donde se evidencia el patrón observador.

- **Strategy (Estrategia):** El patrón strategy define una familia de algoritmos, de tal manera que encapsula cada uno de ellos y los hace intercambiables entre sí. Garantiza total independencia entre las operaciones de los algoritmos y quien los usa. Este patrón se evidencia entre la clase Origen y las clases OrigenPredefinido, OrigenRango y OrigenFormato, que son todos los posibles orígenes de datos con los que permite trabajar la aplicación.

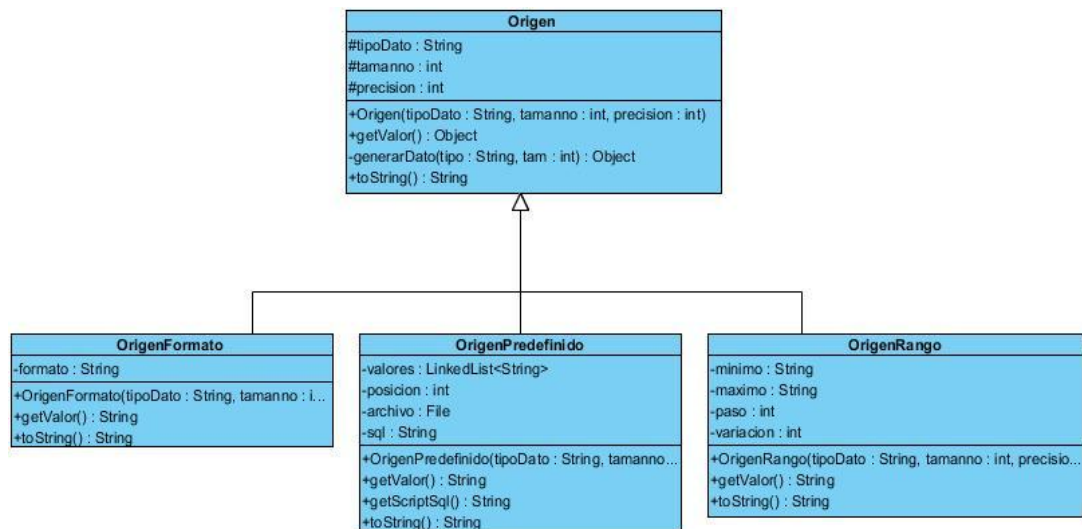


Figura 8 Fragmento del diagrama de clases donde se evidencia el patrón estrategia.

### 2.6 Diseño de la solución

Después de haber dividido las historias de usuario en tareas, deben ser transformadas en código, para esto se llevan a cabo reuniones donde se presentan las tarjetas CRC (Clase Responsabilidad Colaboración), que darán una idea de cuales clases se deben implementar.

#### 2.6.1 Tarjetas CRC

La utilización de tarjetas CRC (Class-Responsibility-Collaboration), del español Colaboración de Responsabilidad de Clases), es una técnica de diseño orientado a objetos propuesta por Kent Beck y Ward Cunningham. El objetivo de la misma es hacer, mediante tarjetas, un inventario de las clases que van a ser utilizadas para implementar el sistema y la forma en que van a interactuar, con ello se pretende facilitar el análisis y discusión de las mismas por parte de varios actores del equipo de proyecto, con el objetivo de que el diseño sea lo más simple posible, verificando las especificación. (Casas, et al., 2011)

A continuación se evidencia la Tarjeta CRC 8 la cual hace referencia a la clase Proyecto, ya que la misma al ser la clase controladora es la encargada de manejar todo el flujo de negocio dentro de la aplicación y contiene los métodos de generación de datos fundamentales para lograr un correcto funcionamiento de la herramienta. ([Ver Anexo](#)) para las restantes tarjetas CRC.

Tarjeta CRC	
Clase: Proyecto	
Responsabilidades	Colaboraciones
Contiene las funcionalidades para generar de las diferentes formas posibles en las tablas de las Bases de Datos.	Editor.java. BaseDatos.java.

Tabla 10 Tarjeta CRC de la clase proyecto.

## 2.6.2 Diagrama de clases

El diagrama de clases contiene información de cómo se establece la relación entre un objeto y otro, la herencia de propiedades de otro objeto y presenta las clases del sistema con sus relaciones estructurales y de herencia. Además son utilizados para modelar la parte estática del software.

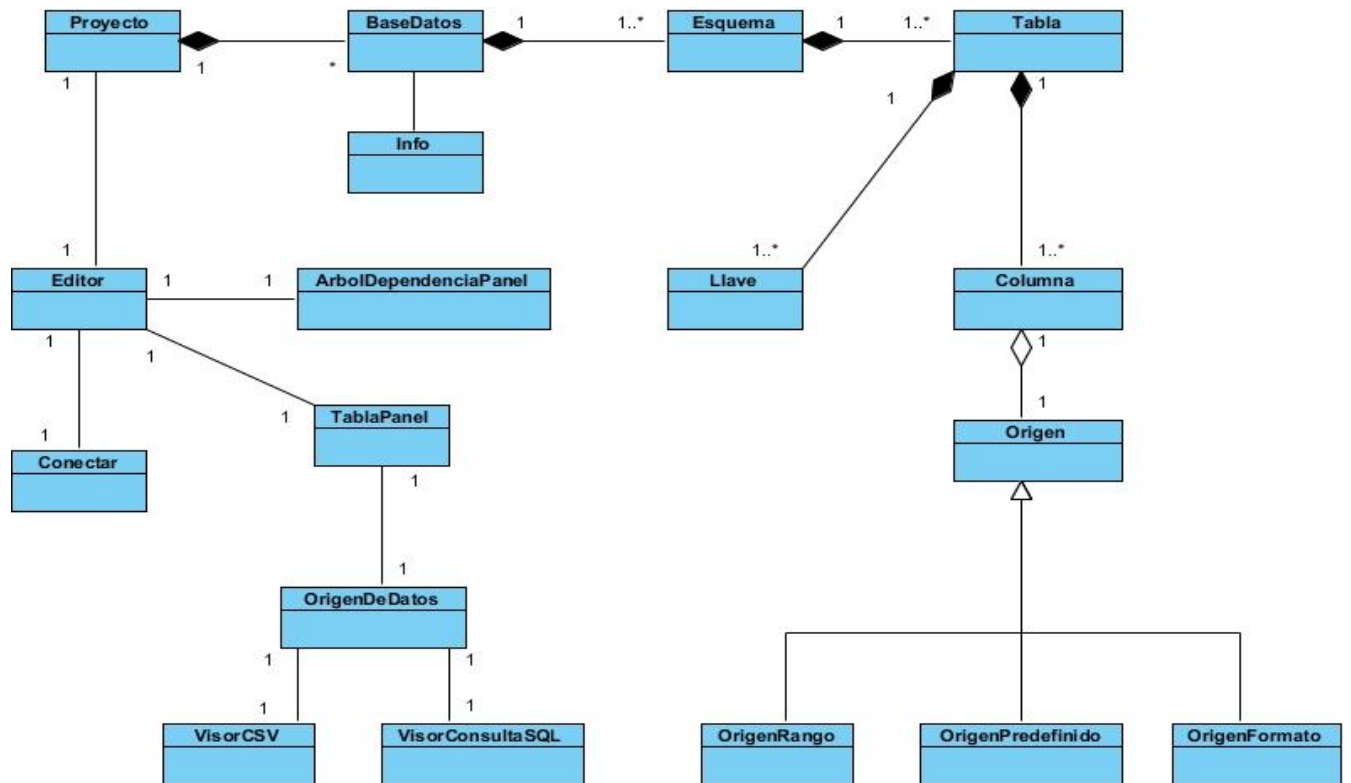


Figura 9 Diagrama de clases.

## 2.7 Conclusiones parciales

Luego del desarrollo del presente capítulo se puede llegar a las siguientes conclusiones:

- Se hizo una breve descripción de la herramienta propuesta y se definieron 26 HU que debe cumplir dicha aplicación.
- Se elaboró el diagrama de clases del negocio con el objetivo de entender mejor la aplicación que se desea desarrollar.

## CAPÍTULO 2: REQUISITOS Y DISEÑO DEL SISTEMA

---

- Fueron generados los artefactos que concibe la metodología XP tales como la lista de reserva del producto, tareas de ingeniería, plan de iteraciones y tarjetas CRC con el fin de documentar cada evento ocurrido en el proceso de desarrollo.
- Se definió el patrón arquitectónico en capas y se hizo uso de los patrones de diseño GRAPS Controlador, Experto, Creador, Alta Cohesión y Bajo Acoplamiento y los patrones GOF Observer y Strategy con el objetivo de lograr una mejor organización de la aplicación.

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

### 3.1 Introducción

Seguido de la fase de exploración y planificación, XP define las fases Iteraciones a primera liberación y producción. En la planificación se definieron las iteraciones y en cada iteración se diseñan, prueban y codifican cada una de las historias de usuario.

En el siguiente capítulo se muestra todo lo relacionado a cada iteración mediante la descripción de elementos en el proceso de implementación. Se visualizan las interfaces de usuario y se da una explicación de cada una de ellas para un mayor entendimiento acerca de la estructura de la aplicación. Se define el estándar de codificación que se estará utilizando para el desarrollo de la solución y por último se ejecutan los niveles de pruebas propuestos por la metodología XP donde se valida el funcionamiento de los requisitos funcionales.

### 3.2 Implementación

Luego de haber definido los elementos necesarios en la etapa de exploración y diseño se pasa a la de codificación o implementación de la aplicación donde se da cumplimiento al Plan de iteraciones. En cada iteración se desarrollan las sub-fases de Diseño, realización de Pruebas unitarias, Codificación de la solución y Refactorización del código. Al finalizar esta fase el cliente estará listo para realizar las Pruebas de aceptación.

#### 3.2.1 Estándares de codificación

La implementación se realizará bajo los estándares de codificación que dictan las convenciones de código para el lenguaje de programación Java.

A continuación cada una de las especificaciones con las que cuenta este estándar de codificación y un ejemplo de cómo se evidencia el mismo dentro de la herramienta de generación de datos.

#### Comentarios

- **Comentarios de una línea:** comentario pequeño que solo abarca una línea y describe el código que le sigue.

#### Declaraciones

- **Cantidad por línea:** es recomendable una sola declaración por línea.
- **Inicialización:** inicializar las variables donde se declaran, a no ser que su valor inicial dependa de algún cálculo o procedimiento.

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

- **Clases e Interfaces:** no debe haber ningún espacio en blanco entre el nombre de un método y el paréntesis “(”. La llave de apertura “{” debe aparecer después de la declaración y la llave de cierre “}” en una nueva línea indentada justo a su sentencia de apertura, excepto cuando no haya código implementado entre ellas.

### Sentencias

- **Sentencias simples:** una sentencia por línea.
- **Sentencias compuestas:** son sentencias que contienen listas de sentencias. La sentencia encerrada entre llave debe estar indentada un nivel más. La llave de apertura “{” debe aparecer al final de la línea de la sentencia compuesta y la de cierre “}” en una línea nueva al nivel de la sentencia compuesta.

### Estilo de los nombres

- **Clases e Interfaces:** los nombres de las clases tendrán la primera letra mayúscula, y si son compuestos la primera letra de cada palabra en mayúscula, nombres simples y de alguna manera que describa el contenido, usar palabras completas, a no ser que la abreviatura sea muy conocida, ej. (UML, BD)
- **Métodos y variables:** en el caso de los métodos será en minúscula, en caso de ser un nombre compuesto la inicial de la primera palabra en minúscula y la de las otras palabras que lo componen en mayúscula. Los nombres de las variables no deben comenzar con los caracteres subguión bajo “\_” o signo de peso “\$”, deben ser cortos pero con significados lógicos, capaz de permitir a un observador identificar su función. (Hommel, et al., 1999)

### 3.2.2 Principales bibliotecas utilizadas

#### Java Database Connectivity (JDBC)

Java Database Connectivity, más conocida por sus siglas JDBC, es una biblioteca que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice. El JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos



## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. (Oracle, 2013)

Entre sus principales ventajas se encuentran la manipulación de cualquier tipo de tareas con la base de datos a las que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos.

En la aplicación se utiliza esta biblioteca de clases a la hora de ejecutar operaciones sobre las bases de datos PostgreSQL, el JDBC permite consultar el catálogo y acceder a cualquier información dentro del mismo (esquemas, tablas, tipos de datos, etc.), además mediante este se realizan todas las operaciones de escritura a la hora de insertar en las tablas los datos generados.

```
25  
26 public boolean conectar(Connection c) throws SQLException, ClassNotFoundException {  
27     Class.forName("org.postgresql.Driver");  
28  
29     return !c.getCatalog().equals("");  
30 }
```

Figura 10 Fragmento de código donde se evidencia la biblioteca JDBC.

### Xeger

Es una biblioteca de Java creada para la generación de cadenas que coinciden con una expresión regular específica. Esta permite generar datos con una semántica de datos correcta, requisito fundamental de la aplicación para que los datos tengan un sentido a la hora de ser insertados en los subsistemas “Contabilidad” y “Costos y procesos”.

```
13
14     @Override
15     public String getValor() {
16         Xeger g = new Xeger(formato);
17         return g.generate();
18     }
19
```

Figura 11 Fragmento de código donde se evidencia el uso la biblioteca xeger.

## 3.3 Interfaces de usuario

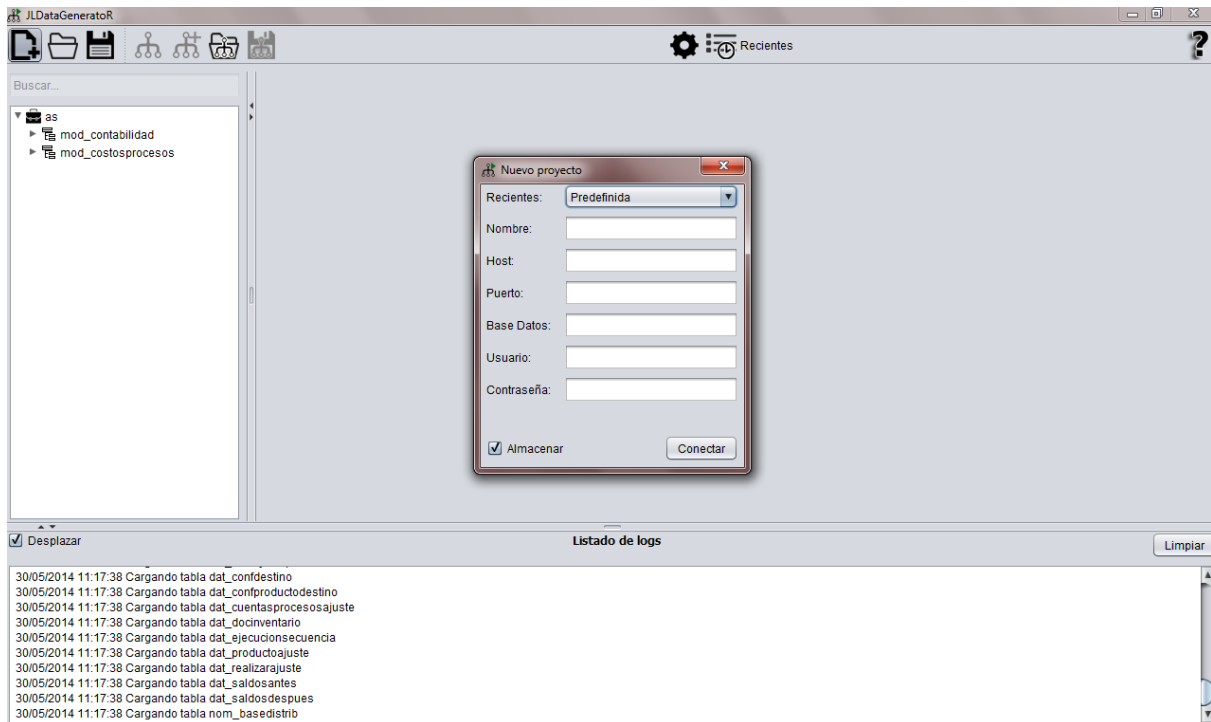


Figura 12 Interfaz de usuario #1 Crear proyecto.

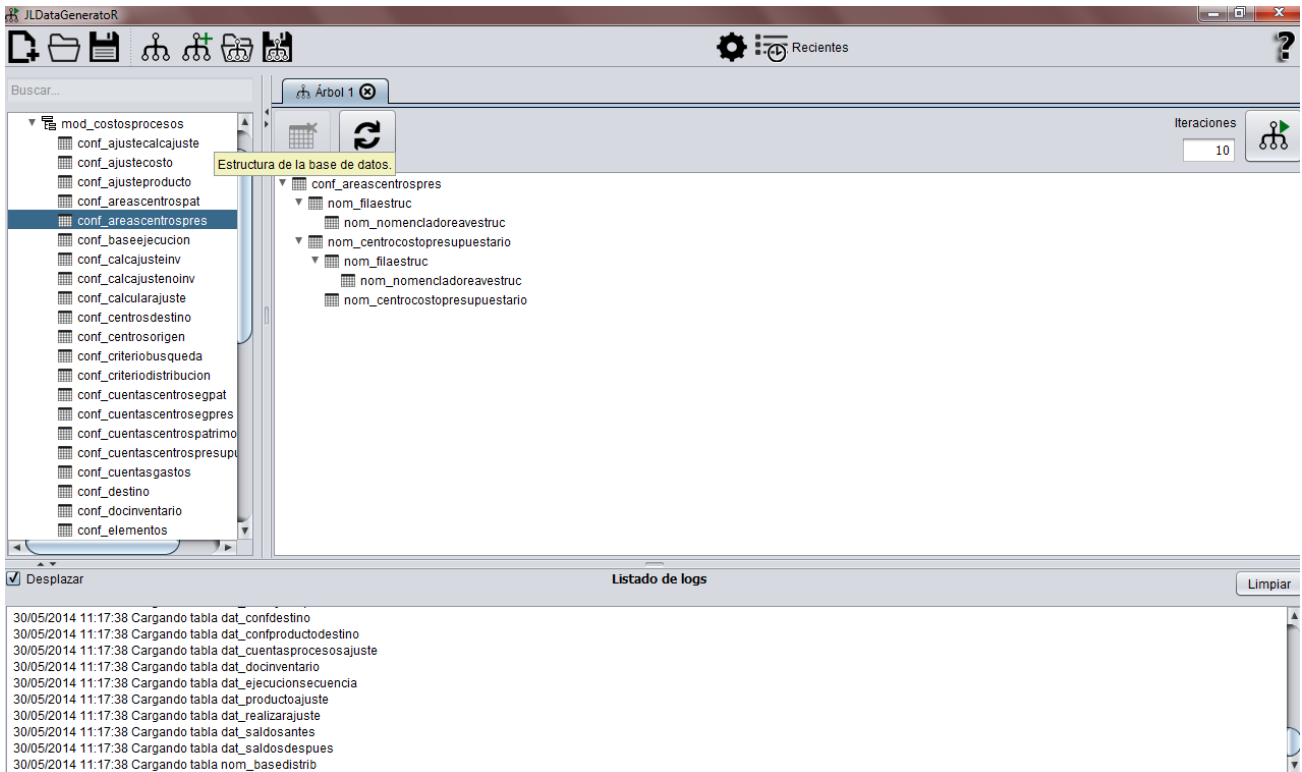


Figura 13 Interfaz de usuario #2 Crear árbol de dependencias.

## 3.4 Validación de requisitos

### Revisión Técnica Formal

El objetivo de la revisión formal es descubrir errores en la función, la lógica o la implementación de cualquier producto del software, verificar que satisface sus especificaciones, que se ajusta a los estándares establecidos, señalando las posibles desviaciones detectadas. Es un proceso de revisión riguroso, su objetivo es llegar a detectar los posibles defectos o desviaciones en los productos que se van generando a lo largo del desarrollo. Esta característica fuerza a que se adopte esta práctica únicamente para productos que son de especial importancia, porque de otro modo podría frenar la marcha del proyecto. (Pressman, 2005)

**Prototipos de interfaz de usuario:** esta técnica permitió la obtención de prototipos a partir de la definición de requisitos que, sin tener la totalidad de la funcionalidad del sistema, permitieron al usuario hacerse una idea de la estructura de la interfaz del sistema. Mediante los prototipos de interfaz de usuario se obtuvo una visión inicial del sistema a implementar demostrando cómo se van a disponer posteriormente los conceptos que intervienen en el mismo. Todo esto permitió corregir la aparición de

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

errores en las fases tempranas del software, evitando gastos innecesarios; sirviendo además como una forma de validar los requisitos funcionales que fueron capturados. (Pressman, 2005)

### 3.5 Verificación del diseño

#### 3.5.1 Tamaño Operacional de Clase (TOC)

A continuación se muestran las clases a las que se les aplicó la métrica TOC.

No	Sistema	Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
1	JRGenerator	BaseDatos	7	Media	Media	Media
2	JRGenerator	Columna	1	Baja	Baja	Alta
3	JRGenerator	Esquema	1	Baja	Baja	Alta
4	JRGenerator	Origen	2	Baja	Baja	Alta
5	JRGenerator	OrigenRango	1	Baja	Baja	Alta
6	JRGenerator	OrigenColumna	3	Baja	Baja	Alta
7	JRGenerator	VisorConsultaSQL	1	Baja	Baja	Alta
8	JRGenerator	VisorCSV	2	Baja	Baja	Alta
9	JRGenerator	Editor	18	Alta	Alta	Baja
10	JRGenerator	OrigenDatos	3	Baja	Baja	Alta
11	JRGenerator	ArbolDependencia Panel	7	Media	Media	Media
12	JRGenerator	Llave	4	Baja	Baja	Alta

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

13	JRGenerator	OrigenPredefinido	3	Baja	Baja	Alta
14	JRGenerator	OrigenFormato	1	Media	Media	Media
15	JRGenerator	Proyecto	12	Media	Media	Media
16	JRGenerator	Tabla	8	Media	Media	Media

Tabla 11 Clases a las que se les aplicó la métrica TOC.

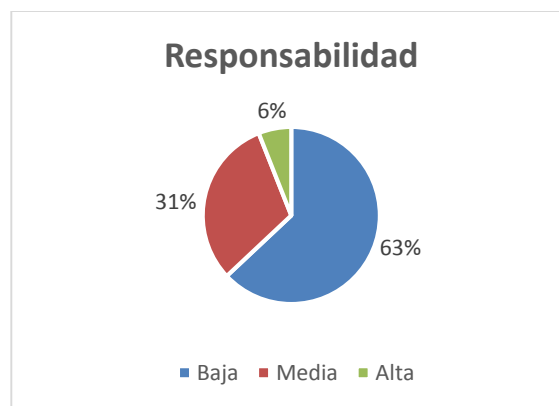


Figura 14 Resultados de la métrica TOC. Atributo: Responsabilidad.

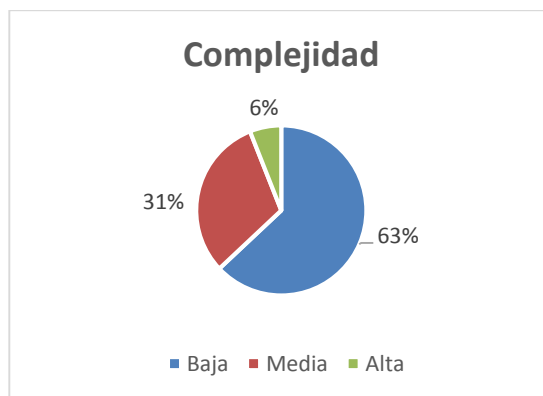


Figura 15 Resultados de la métrica TOC. Atributo: Complejidad.

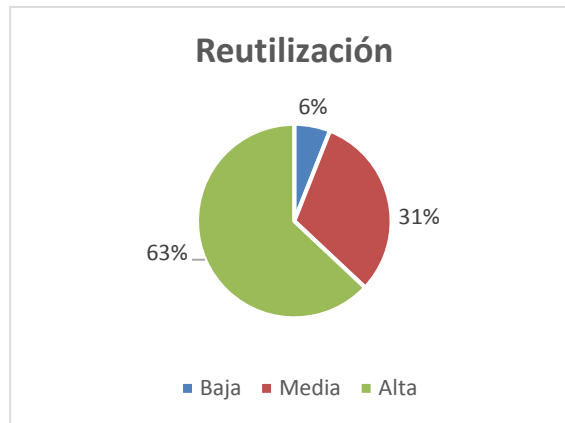


Figura 16 Resultados de la métrica TOC. Atributo: Reutilización.

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC en la evaluación del instrumento, se puede observar que la mayoría de las clases que conforman el sistema para los atributos de calidad responsabilidad y complejidad, están dentro de la categoría Media y Baja para un 94% del total, mientras que el atributo Reutilización cuenta con igual por ciento en las categorías Alta y Media mostrando así que el componente cuenta con una elevada reutilización, baja complejidad y responsabilidad en el diseño propuesto. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

### 3.5.2 Relaciones entre Clases (RC)

La tabla que se muestra a continuación ofrece las 18 clases del diagrama del diseño a las que se les aplicó la métrica RC.

No	Sistema	Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad de Mantenimiento	Reutilización	Cantidad de Pruebas
1	JRGenerator	Proyecto	2	Medio	Baja	Alta	Baja
2	JRGenerator	BaseDatos	3	Alto	Media	Media	Media

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

3	JRGenerat or	Esquema	2	Medio	Baja	Alta	Baja
4	JRGenerat or	Tabla	3	Alto	Media	Media	Media
5	JRGenerat or	Editor	4	Alto	Media	Media	Media
6	JRGenerat or	ArbolDepe ndenciasP anel	1	Bajo	Baja	Alta	Baja
7	JRGenerat or	TablaPanel	2	Medio	Baja	Alta	Baja
8	JRGenerat or	Conectar	1	Bajo	Baja	Alta	Baja
9	JRGenerat or	Llave	1	Bajo	Baja	Alta	Baja
10	JRGenerat or	Columna	2	Medio	Baja	Alta	Baja
11	JRGenerat or	Origen	4	Alto	Media	Media	Media
12	JRGenerat or	OrigenDato s	3	Alto	Media	Media	Media
13	JRGenerat or	VisorCSV	1	Bajo	Baja	Alta	Baja

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

14	JRGenerat or	VisorConsu ltaSQL	1	Bajo	Baja	Alta	Baja
15	JRGenerat or	OrigenRan go	1	Bajo	Baja	Alta	Baja
16	JRGenerat or	OrigenColu mna	2	Medio	Baja	Alta	Baja
17	JRGenerat or	OrigenPred efinido	1	Bajo	Baja	Alta	Baja
18	JRGenerat or	OrigenFor mato	1	Bajo	Baja	Alta	Baja

Tabla 12 Clases del diagrama del diseño a las que se les aplico la métrica RC.

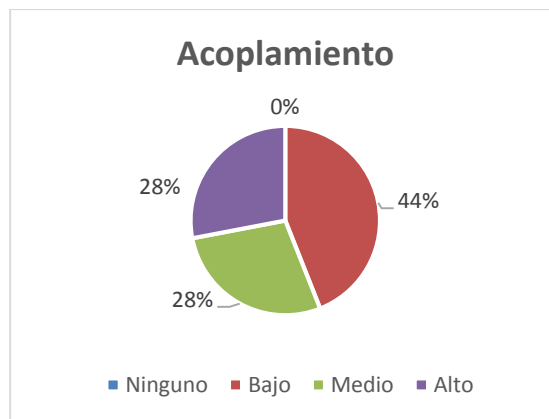


Figura 17 Resultados de la métrica RC. Atributo: Acoplamiento.



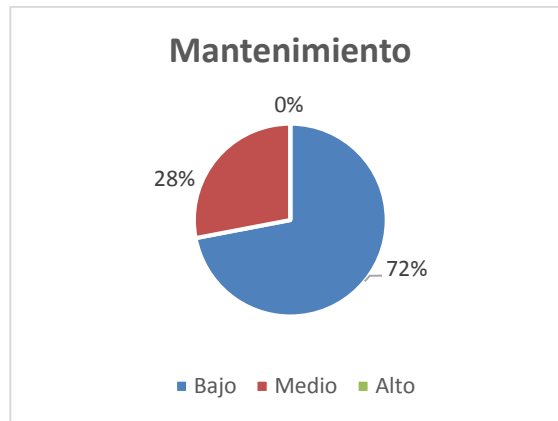


Figura 18 Resultados de la métrica RC. Atributo: Mantenimiento.

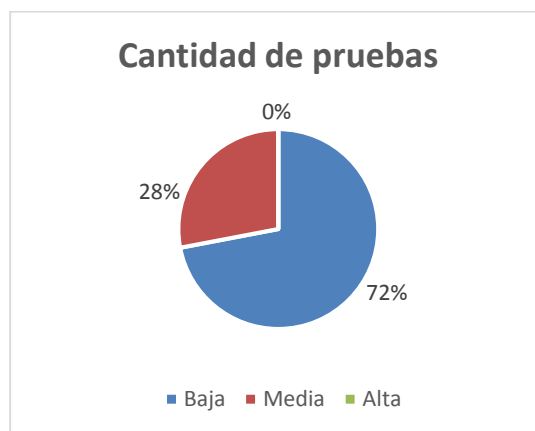


Figura 19 Resultados de la métrica RC. Atributo: Cantidad de pruebas.

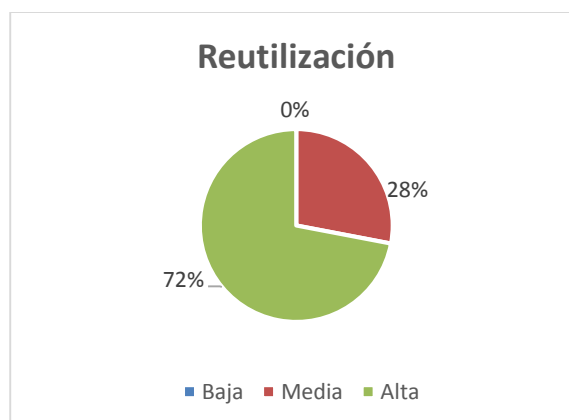


Figura 20 Resultados de la métrica RC. Atributo: Reutilización.

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que las clases del diseño posee un bajo acoplamiento, ya que para este atributo las categorías ninguno y bajo sumaron un 44% del total, mostrando un 72% en la categoría alta del atributo reutilización. Los atributos complejidad de mantenimiento y cantidad de pruebas, sumaron el mismo por ciento en las categorías baja y media, lo que demuestra que no es necesario un elevado esfuerzo en el momento de realizar cambios, rectificaciones y pruebas al software.

### **3.6 Pruebas software**

Cuando se desarrolla una solución informática se deben realizar una gran cantidad de pruebas para verificar que el código esté correcto. Estas pruebas normalmente tienen que ser ejecutadas en varias ocasiones y se ven afectadas por los cambios que se introducen conforme se va construyendo la solución. XP divide las pruebas en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente final.

#### **3.6.1 Pruebas unitarias**

Las pruebas de unitarias son una forma de probar el correcto funcionamiento del código. Esto sirve para asegurar que cada uno funcione correctamente por separado. Para el desarrollo de las pruebas de la aplicación se seleccionaron los métodos más relevantes. Se utilizó la herramienta JUnit para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado. Además debido a su compatibilidad con el lenguaje Java. A continuación se muestran los casos de pruebas creados:

# CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

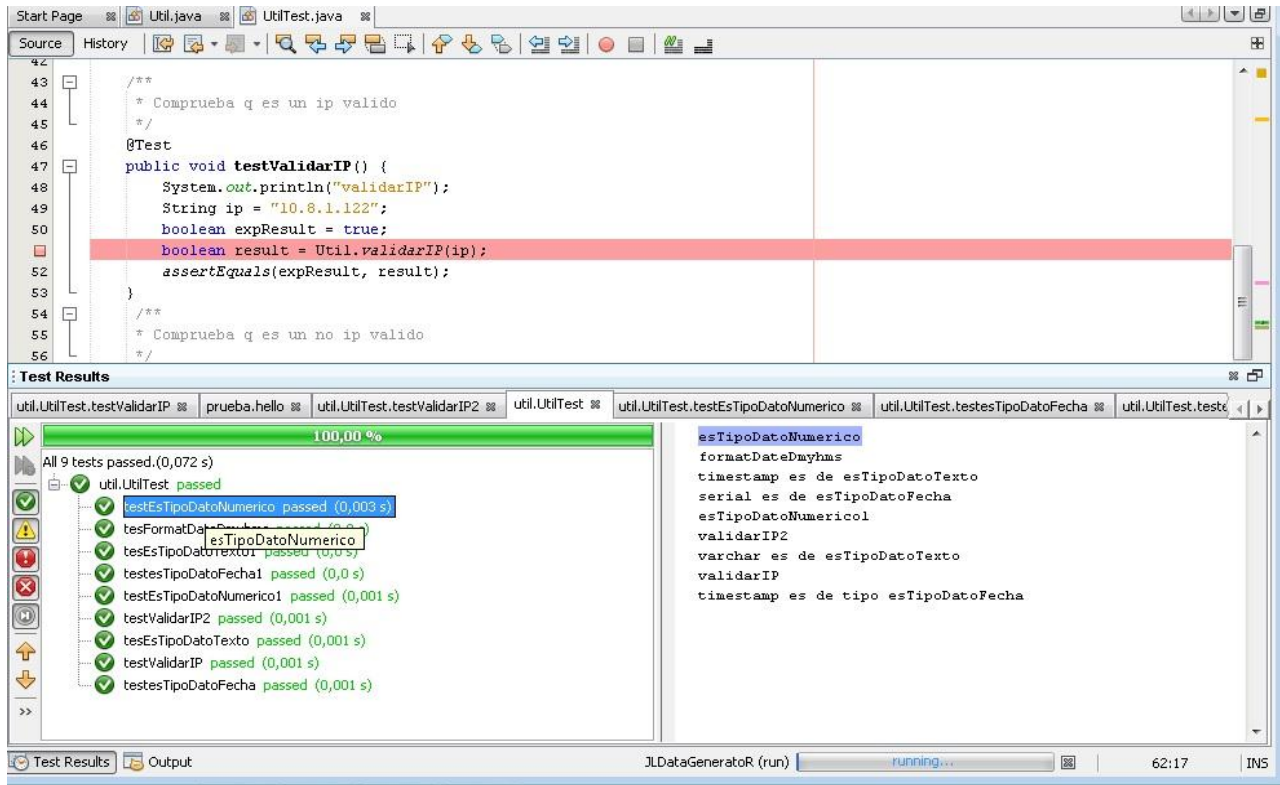


Figura 21 Pruebas unitarias #1 .

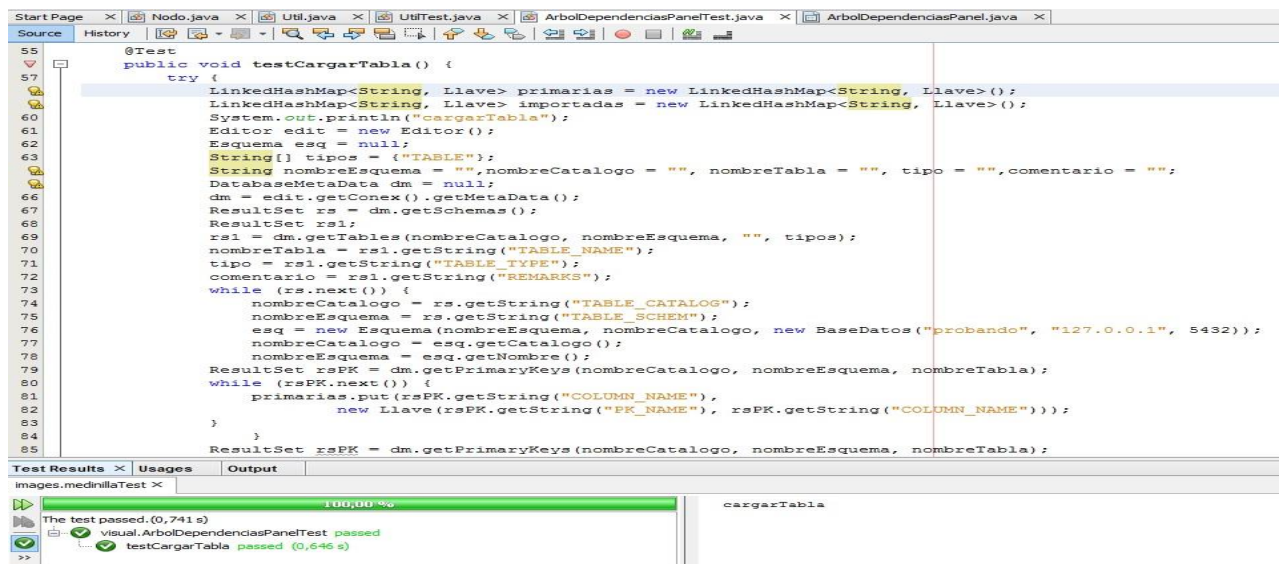


Figura 22 Pruebas unitarias #2.

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

Después de aplicadas las pruebas con la herramienta JUnit se puede concluir que el 100% de las clases fueron ejecutadas correctamente. Por lo que se evidencia que el código se ejecuta de manera positiva sin ningún tipo de errores.

### 3.6.2 Pruebas de aceptación

Las pruebas de aceptación XP, también llamadas pruebas del cliente son especificadas por el mismo y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Estas pruebas derivan de las HU que se han implementado como parte de la liberación del software. (Joskowicz, 2008)

Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información. (Joskowicz, 2008)

A continuación se muestra cómo se precisaron las pruebas de aceptación, las cuales se encuentran separadas en casos de pruebas:

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1:Generar datos mediante un formato	Se generan datos mediante un formato definido por el usuario	EP 1.1 Generar datos mediante un formato y se presiona el botón ejecutar.	Se selecciona una tabla del <b>Árbol de dependencias</b> .  Se presiona <b> doble clic</b> o <b> clic derecho</b> sobre la tabla para <b> Ver detalles</b>  Se presiona <b> doble clic</b> sobre una fila de la tabla.  Se selecciona la opción <b> Según tipo de dato</b> .

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

			<p>Se selecciona la opción <b>Formato</b>.</p> <p>Se escriben los valores en campo de texto.</p> <p>Se presiona el botón <b>Guardar</b>.</p> <p>Se selecciona la cantidad de iteraciones a generar.</p> <p>Se presiona el botón <b>Ejecutar</b>.</p> <p>.</p>
		EP 1.2 Generar datos mediante un formato y se presiona el botón Guardar.	<p>Se selecciona una tabla del <b>Árbol</b> de dependencias.</p> <p>Se presiona doble clic <b>o</b> clic derecho sobre la tabla para <b>Ver detalles</b></p> <p>Se presiona doble clic sobre una fila de la tabla.</p> <p>Se selecciona la opción <b>Según tipo de dato</b>.</p> <p>Se selecciona la opción <b>Formato</b>.</p> <p>Se escriben los valores en el campo de texto.</p> <p>Se presiona el botón Guardar.</p>

		<p>EP 1.3 Generar datos mediante un formato y se presiona el botón Guardar sin elementos en el campo de texto.</p>	<p>Se selecciona una tabla del <b>Árbol</b> de dependencias.</p> <p>Se presiona doble clic o clic derecho sobre la tabla para <b>Ver detalles</b>.</p> <p>Se presiona doble clic sobre una fila de la tabla.</p> <p>Se selecciona la opción <b>Según tipo de dato</b>.</p> <p>Se selecciona la opción <b>Formato</b>.</p> <p>Se presiona el botón Guardar.</p> <p>Se muestra un mensaje de error.</p>
--	--	--	---

Figura 23 Diseño de clases de prueba para la historia de usuario Generar datos mediante un formato.

### Condiciones de ejecución

- 1 Tiene que haberse creado un proyecto previamente.
- 2 Tiene que haberse creado un árbol de dependencias

### 3.6.3 Resultados de pruebas

Para validar el correcto funcionamiento de la aplicación y en correspondencia con lo que plantea la metodología XP, se realizaron 3 iteraciones de pruebas, dos al finalizar cada una de las iteraciones definidas en el plan de desarrollo, con el objetivo de probar el correcto funcionamiento de las funcionalidades implementadas dentro de las mismas; y la otra al finalizar el desarrollo de la aplicación, en aras de probar el comportamiento de la herramienta una vez integradas todas las funcionalidades implementadas en las dos iteraciones de desarrollo planificadas. Como resultado de estas pruebas en una primera iteración se encontraron en total 12 no conformidades: 6 de validación, 2 de ortografía y 4

## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

de interfaz. En la segunda iteración fueron encontradas 4 no conformidades: 2 de validación, 1 de ortografía, 1 de interfaz. En la 3ra iteración se obtuvieron resultados satisfactorios, encontrándose 0 no conformidades. Antes de pasar a una nueva iteración de prueba se realizaron pruebas de regresión con el objetivo de verificar que las no conformidades encontradas se habían resuelto o que los cambios realizados no habían provocado nuevas no conformidades. Ver figura:

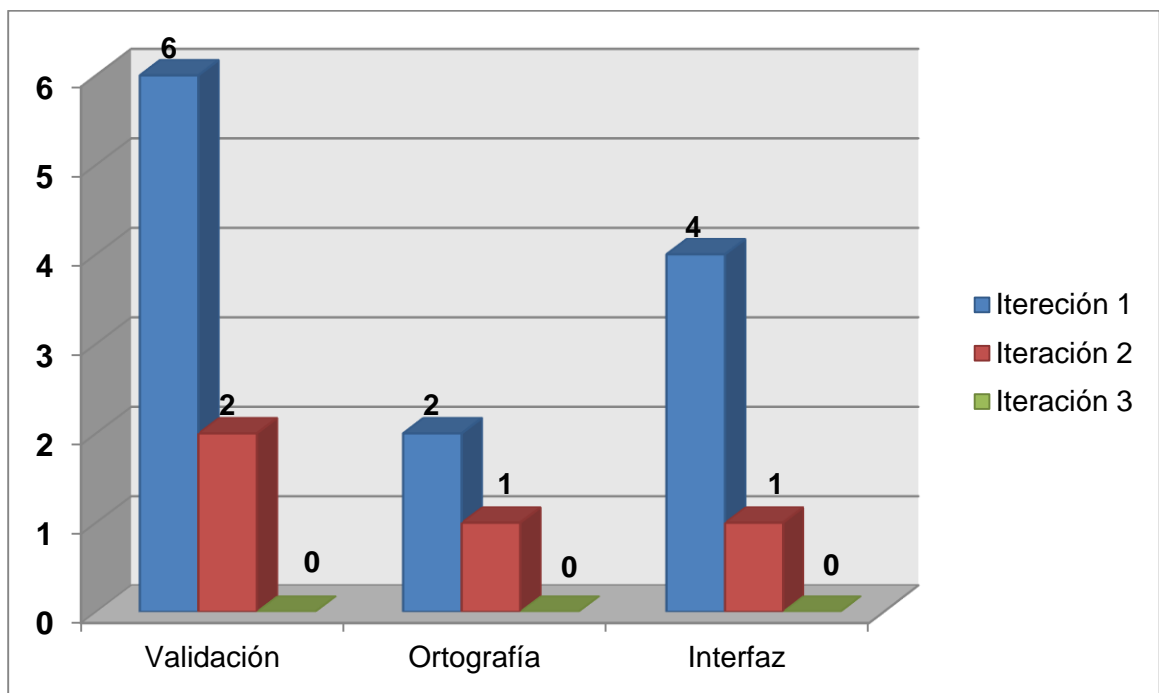


Figura 24 Resultados de prueba.

### 3.6.4 Pruebas de usabilidad

Las pruebas de usabilidad es una técnica usada en el diseño de interacciones centrado en el usuario para evaluar un producto mediante pruebas con los usuarios mismos. Esto puede ser visto como una práctica de usabilidad irremplazable, dado que entrega información directa de como los usuarios reales utilizan el sistema. (Nielsen, 1994)

Para la validación del requisito no funcional usabilidad se utilizó la lista de chequeo establecida en el centro CEIGE por una tesis de maestría realizada en este propio centro en el año 2012, teniendo en cuenta los principios de la heurística de la usabilidad. El siguiente gráfico muestra el comportamiento en % de los principios:

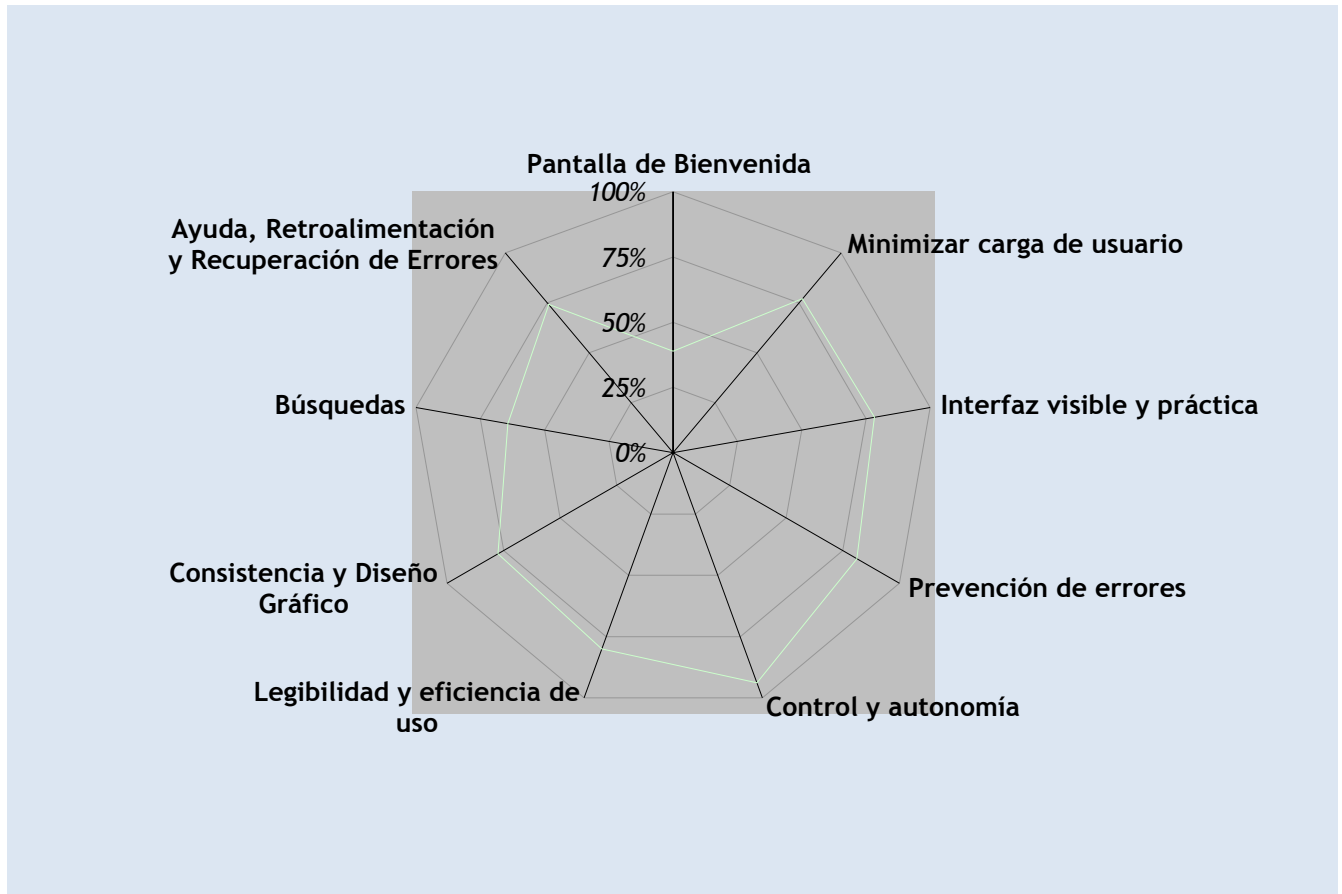


Figura 25 Principios de la heurística de usabilidad

Según el estándar ISO 14598 para que un requisito de usabilidad sea clasificado como satisfactorio, al realizar las pruebas utilizando las listas de chequeo el grado de usabilidad debe de ser superior al 60%. Al aplicar la lista de chequeo entre los principales desarrolladores del centro CEIGE que serán los encargados de trabajar con la herramienta, se obtuvo como resultado un grado de Usabilidad de un 74%, lo que demuestra que la herramienta cuenta con un elevado grado de usabilidad.

### 3.7 Validación de la variable de la investigación

Con el objetivo de demostrar la eficiencia de la herramienta de generación de datos para los subsistemas “Contabilidad y Costos y Procesos” se realizó un experimento en conjunto con un especialista del Departamento de Contabilidad.

El experimento consistió en la definición de un caso de estudio que contó con dos escenarios de desarrollo y la variable tiempo como criterio de comparación. En el primer escenario el especialista



## CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

---

realizó el proceso de carga de datos tal y como se realiza actualmente, ejecutándolo de forma manual al nomenclador cuenta del subsistema contabilidad, teniendo en cuenta que los nomencladores asociados a él contenían los datos necesarios para el registro de las nuevas cuentas. Este proceso fue estimado en un tiempo de 2 horas para un total de 60 cuentas. En el segundo escenario se utilizó la herramienta implementada, para ello fueron generados los datos de los nomencladores asociados al nomenclador cuenta, cuya tabla homóloga con la base de datos es nom\_cuenta del esquema contabilidad, la cual se relaciona con 12 tablas generando 10 tuplas por cada una utilizando la técnica de generación por una lista de valores definidos por el usuario para configurar los campos de las tablas. Con el fin de generar 60 tuplas de la tabla nom\_cuenta se utilizó la técnica generar datos a través de una consulta para obtener los valores de las dependencias antes generadas y la técnica de generar datos mediante un formato para insertar datos con la mayor semántica posible, este proceso se demoró 17 minutos. Demostrando que la herramienta implementada mejora el tiempo de ejecución en un 85%.

### 3.8 Conclusiones parciales

Luego del desarrollo del presente capítulo se puede arribar a las conclusiones siguientes:

- Se decidió utilizar el estándar de código definido para el lenguaje java en el desarrollo de la aplicación, con el objetivo de lograr un mejor entendimiento del código y para permitir un mejor mantenimiento del mismo.
- Se realizaron pruebas de aceptación y pruebas unitarias lo que permitió detectar, documentar y corregir las no conformidades existentes en el sistema implementado.
- Al concluir el período de pruebas se obtuvo una aplicación que cumple de forma correcta con la totalidad de las funcionalidades esperadas por el cliente.

### CONCLUSIONES GENERALES

Al finalizar la investigación del presente trabajo se puede concluir que:

- Al realizar el análisis se obtuvieron 26 HU de las que se identificaron 26 requisitos funcionales, los cuales permitieron conocer la manera en el que el sistema debe comportarse ante situaciones particulares, en el caso del presente trabajo, la necesidad de poblar bases de datos desarrolladas en PostgreSQL con información semánticamente correcta.
- Se obtuvieron durante el diseño de la herramienta 18 tarjetas CRC así como 18 clases que conformaron el diagrama de clases, obteniendo con ello una visión global de la estructura del sistema, así como la relación entre sus elementos, propiciando la incorporación de nuevos elementos como patrones o estilos que enriquecieran el resultado final.
- Se implementó una herramienta para la generación de datos, la cual ayudará en el proceso de prueba y presentación a potenciales clientes de los subsistemas “Contabilidad y Costo de Procesos” pertenecientes al proyecto Xedro-ERP, gracias a que permite disminuir el tiempo mediante la puesta en práctica, para su desarrollo, de técnicas avanzadas de programación adquiridas durante la carrera.
- Las pruebas de aceptación aplicadas para la validación de la herramienta, demostraron que el sistema cumple con los requisitos definidos, garantizando de esta forma el cumplimiento del objetivo general de este trabajo.
- Por último, el resultado de este trabajo tendrá un gran impacto: desde el punto de vista económico en el ahorro de tiempo en los procesos vinculados con la presentación a potenciales clientes y pruebas de software; y desde el punto de vista social humanizando los antes mencionados procesos al brindar una herramienta de apoyo capaz de realizar las actividades de población de bases de datos en PostgreSQL en pocos pasos.

### RECOMENDACIONES

- Permitir que la herramienta genere datos para otros gestores de bases de datos relacionales como: MySQL, SQL Server u Oracle.
- Incluir en la herramienta el soporte para otros tipos de datos que no están presente en el Xedro-ERP con el objetivo de lograr una herramienta más completa; por ejemplo: money, xml.

### BIBLIOGRAFÍA REFERENCIADA

[En línea] [Citado el: 3 de Abril de 2014.] <http://synergix.wordpress.com/2008/07/10/modelo-de-dominio/>.

**Entornos de programación. 2012.** Entornos de Programación. *Entornos de Programación*. [En línea] 17 de 1 de 2012. [Citado el: 2 de Marzo de 2014.] <http://lml.ls.fi.upm.es/ep/entornos.html#toc5>.

**Álvarez, Sara. 2007.** Desarrollo Web. [En línea] 2007. [Citado el: 24 de Febrero de 2014.] <http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html>.

**Carga de Datos. 2014.** Carga de Datos. [En línea] 2014. [Citado el: 25 de Febrero de 2014.] [http://200.43.105.174/tsas/common/themes/tsas/help/Carga\\_de\\_Datos.htm](http://200.43.105.174/tsas/common/themes/tsas/help/Carga_de_Datos.htm).

**Casas, Sandra y Reinaga, Hector. 2011.** *Identificación y Modelado de Aspectos Tempranos dirigidos por Tarjetas de Responsabilidades y Colaboraciones*. 2011.

**Cobo, Angel Yera.** *Diseño y programación de bases de datos*. Madrid : Vision Libros. 978-84-9821-459-8.

**Connexions. 2008.** El Glosario IEEE de Ingeniería del Software. [En línea] 2008. [Citado el: 24 de Febrero de 2014.] <http://cnx.org/content/m17423/latest/>.

**Datanamic. 2014.** Datanamic. [En línea] Datanamic Solutions BV, 2014. [Citado el: 26 de Febrero de 2014.] <http://www.datanamic.com/datagenerator/index.html>.

**Date, Christopher J. 2001.** *Introducción a los Sistemas de Bases de Datos*. 2001. Séptima Edición.

**EMS. 2014.** SQL Manager. *SQL Manager*. [En línea] EMS Database Management Solutions, 2014. [Citado el: 26 de Febrero de 2014.] <http://www.sqlmanager.net/products/postgresql/datagenerator>.

**González, Miguel Angel. 2012.** Prezi. [En línea] Herramientas CASE, 13 de Septiembre de 2012. [Citado el: 26 de Marzo de 2014.] [http://prezi.com/aad8mbta\\_vjb/herramientas-case](http://prezi.com/aad8mbta_vjb/herramientas-case).

**Hommel, Scott y Molpeceres, Alberto. 1999.** (Convenciones de Código para el lenguaje de programación JAVA™. *Traducido al castellano 10 Mayo del 2001* . [En línea] Sun Microsystems Inc., 20 de Abril de 1999. [Citado el: 14 de Marzo de 2014.] <http://www.javahispano.com>.

**JavaHispano. 2014.** JavaHispano. [En línea] 2014. [Citado el: 2 de Marzo de 2014.] <http://www.javahispano.org/>.

**Joskowicz, Jose. 2008.** Reglas y Prácticas en eXtreme Programming. [En línea] 2008. [Citado el: 28 de Febrero de 2014.] <http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf>.

- Korel, Bogdan. 1990.** *Automated Software Test Data Generation*. s.l. : IEEE Transactions on Software Engineering, 1990.
- Larman, Craig. 1999.** *UML y Patrones Introducción al análisis y diseño orientado a objetos*. Mexico : s.n., 1999.
- Letelier, Patricio y Penadés, Maria Carmen. 2006.** *Métodologías ágiles para el desarrollo de software:eXtreme Programming (XP)*. Universidad Politécnica de Valencia : s.n., 2006.
- Lorenz, Mark y Kidd, Jeff. 1994.** *Object-Oriented Software Metrics*. Englewood Cliffs, Nueva Jersey : Prentice Hall Object-Oriented, 1994.
- Martinez, Rafael. 2009.** PostgreSQL-es. [En línea] 17 de Julio de 2009. [Citado el: 2014 de Febrero de 26.] <http://www.postgresql.org/es/node/313>.
- Netbeans. 2014.** Netbeans. [En línea] 2014. [Citado el: 2 de Marzo de 2014.] <https://netbeans.org/>.
- Nielsen, J. 1994.** *Usability Engineering*. s.l. : Academic Press Inc, 1994.
- OMG. 2014.** OMG. [En línea] OMG, 27 de 2 de 2014. [Citado el: 1 de Marzo de 2014.] [http://www.omg.org/gettingstarted/what\\_is\\_uml.htm](http://www.omg.org/gettingstarted/what_is_uml.htm).
- Oracle. 2013.** Oracle. [En línea] 2013. [Citado el: 2 de Abril de 2014.] <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>.
- Pelaez, Juan. 2009.** *Arquitectura basadas en capas*. [En línea] 9 de Mayo de 2009. [Citado el: 10 de marzo de 2014.] <http://www.juanpelaez.com/geek-stuff/arquitectura/arquitectura-basada-en-capas/>.
- Pérez, C.J Martín. 2010.** PostgreSQL. [En línea] SoftwareLibre, 2010. [Citado el: 27 de Febrero de 2014.] <http://softwarelibre.org/sites/default/files/articulo.pdf>.
- Pérez, Teresa Garzón. 2010.** *Sistemas Gestores de Bases de Datos*. 2010. 1988-6047.
- Pressman, Roger S. 2005.** *Ingeniería de Requisito: Un enfoque práctico*. 2005.
- . **2006.** *Ingeniería del software. Un enfoque práctico*. s.l. : sexta edición, 2006.
- Programación, Lenguajes de. 2009.** Lenguajes de Programación. [En línea] 2009. [Citado el: 1 de marzo de 2014.] <http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml>.
- Rioseco, Marcelo y Samuel, Marjorie. LinkedIn Corporation. 2014.** *Procesamiento de Datos*. [En línea] Procesamiento de Datos, 2014 de LinkedIn Corporation. [Citado el: 24 de Febrero de 2014.] <http://www.slideshare.net/edupa01/procesamiento-de-datos-4880926>.
- Riviera, Bestard y Quiala, Fonseca. 2010.** *Generador de datos para PostgreSQL.Postgen*. Santiago de Cuba : s.n.

**Semagix. 2009.** *SEMAG. The magic of semantic search.* [En línea] 2009. [Citado el: 25 de Febrero de 2014.] [www.semagix.com/what-is-semantic-data.htm](http://www.semagix.com/what-is-semantic-data.htm).

**Software, Unidad Docente de Ingeniería del. 2010.** *Patrones del "Gang of Four".* Facultad de informática - Universidad Politécnica de Madrid : s.n., 2010.

**Vázquez, Pedro Jesús Escudero, García Moreno, María N y García Peñalvo, Francisco J. 2004.** *Métricas orientadas a objetos.* Salamanca : s.n., 2004.

**Visual Paradigm. 2014.** Visual Paradigm. *Visual Paradigm.* [En línea] 12 de 2 de 2014. [Citado el: 2 de 3 de 2014.] <http://www.visual-paradigm.com/>.