Facultad 3

Desarrollo de las funcionalidades para el diseño visual de consultas y archivos de datos en la herramienta ORMaping

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

Autor:

Yoandrys León Batista

Tutor:

Ing. Manuel Álvarez Alonso

Co-Tutor:

Ing. Yeisel Pérez Rivera

Junio, 2016

"Año 56 del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 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 _____.

_____	_____
Yoandrys León Batista	Ing. Manuel Alvarez Alonso
Autor	Tutor

AGRADECIMIENTOS

No es tarea fácil reconocer en pocas palabras a tantas personas que de una forma u otra han influido en mi vida brindándome apoyo, amistad, amor y regalándome buenos y malos momentos. Muchas veces sin ellos saberlo se convirtieron en factores decisivos en mi formación y por eso pienso que este logro no pertenece solo a mí, sino a todas estas personas que han llegado a ocupar un lugar importante en mi vida. Por eso agradezco a:

A mi familia en general. Aunque no puedo dejar de mencionar aquellas personas que por razones ajenas a su voluntad no pueden estar presente hoy aquí:

A mi padre, Jorge Luis León, por haber confiado siempre en mí y por respetar y apoyar mis decisiones.

A mi madre, María Batista, que siempre ha sabido cuidar de mí, darme todo su amor.

A mi segunda madre Deysi y a mi abuela Georgina.

Gracias lo que soy se lo debo a ustedes, han sido mi apoyo en los momentos difíciles. Me han enseñado los mejores valores, gracias a estos he podido andar con paso firme y seguro. Les agradezco por haberme hecho una persona de bien, por las horas de desvelo y dolor que alguna vez pude haberles causado, por mantenerse a mi lado.

A mis hermanos que los quiero mucho.

A mi tutor.

A los miembros del tribunal por cada una de sus sugerencias y recomendaciones.

Al colectivo de profesores de mi facultad en especial al profe Leandro.

A mis amigos de la uci, la gente de la 5 Enrique, laulau, Yanet, Luis Orlando, José Manuel, Dayron, José Carlos, al Pedri, a Tony, a David a la gente de la 3 Roly, Jennifer, Yordan, Raciel, Andres, a Ivaniet, a Yeisel por no volverse loca con mi documento y por toda la ayuda prestada, al Miche por estar siempre presente y brindarme su apoyo incondicional y a mi amiguita Diana por ser tan especial.

A esas personas que, aunque la vida no permitió que tuviéramos los mimos padres se han convertido en mis verdaderos hermanos, formando la familia de los ¨C¨ a GC RanieL, al C Alexei, y SC Kelly gracias a todos ellos por soportar todas mis malcriadeces y gracias por todos los momentos especiales que compartimos y seguiremos compartiendo juntos.

A una personita especial que tuve la posibilidad de conocer en este año y en estos momentos constituye una de las personas que más quiero y aprecio en este mundo a ti Amanda María gracias por ser como eres.

A una amiga muy querida con la cual compartí gran parte de mi vida y tuve la posibilidad de conocer a mi segunda familia, con la que viví buenos y malos momentos, inolvidables todos, a ti Irina gracias por todas las cosas bellas que me regalaste.

A mi gente de lab, Vladimir por todas las batallas compartidas, Yesenia por toda su sinceridad, por alegrarnos los momentos de estrés con sus shows y sus discretas exclamaciones, Ester, Verónica, al Leo los quiero mucho a todos.

DEDICATORIA

Dedico este especial momento a:

Mis padres.

Mis hermanos Yoandra, Yunior, Katerine y Katy

A mi abu por todos sus consejos y enseñanzas.

En especial a una persona que he admirado y querido toda mi vida, más que nada por sus principios y sus valores, por apoyarme en los momentos difíciles, por todos los alones de pelo que me dio y todas sus muestras de cariño, por llegar a significar como esa otra madre que tengo, con la que puedo contar para todo lo que necesite, a Deysi Tamayo labrada gracias por todas las cosas lindas que me diste y gracias por ser tan especial.

RESUMEN

El auge de la informatización de los procesos cobra cada día mayor fuerza, fundamentalmente en nuestro país. Basado en la mejora de la calidad y la eficiencia en la producción, se desarrollan productos informáticos capaces de garantizar el cumplimiento eficaz de los planes económicos y políticos del gobierno cubano, reflejados en los lineamientos del Sexto Congreso del Partido Comunista de Cuba que convergen a la integración de nuestras necesidades con el crecimiento mundial en esta rama.

La Universidad de las Ciencias Informáticas cuenta con el Centro de Gobierno Electrónico (CEGEL), que desarrolla soluciones informáticas para satisfacer las emergentes demandas del mercado y entre sus productos cuenta con ORMaping. El presente trabajo tiene como objetivo el desarrollo de funcionalidades que permitan la realización visual de consultas y archivos de datos, para disminuir el tiempo de realización de las tareas de los arquitectos de datos del centro, posibilitando de esta forma que el desarrollo de consultas a los datos implique menos tiempo y/o personal calificado; además, la existencia de un mecanismo automatizado para la inclusión de datos ya existentes en otras bases de datos, en los archivos de datos del ORM Doctrine, acarreando un mínimo de costo de tiempo de desarrollo por este concepto.

El principal resultado que se alcanza con la realización de la investigación, es una herramienta informática que contribuye a disminuir el tiempo invertido de los arquitectos de datos del centro, en el proceso de realización visual de consultas y archivos de datos, comprobado mediante la validación del sistema a través de las pruebas aplicadas por los especialistas en el tema. Estos resultados fueron obtenidos con el uso de un grupo de herramientas y tecnologías libres como el Entorno de Desarrollo Integrado Netbeans y el lenguaje de programación Java, además de la guía ofrecida por la metodología de desarrollo ágil Extreme Programming.

PALABRAS CLAVE

Archivos de datos, Consulta, Bases de Datos, ORMaping.

Índice de contenidos

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA……...16

1. Introducción 16

1.1. Conceptos fundamentales asociados al dominio del problema 16

1.1.1. Mantenimiento de Software 16

1.1.2. Análisis de ORMaping..... 17

1.1.3. Doctrine..... 19

1.1.4. Datos de prueba 20

1.2. Metodología, lenguajes y herramientas de desarrollo 20

1.2.1. Metodología de desarrollo..... 21

1.2.2. Lenguaje de modelado y herramienta case 24

1.2.3. Lenguaje de programación y entorno de desarrollo integrado 24

1.2.4. Entorno de desarrollo integrado (IDE)..... 25

1.3. Sistema gestor de bases de datos (SGBD) 26

1.4. Bibliotecas adicionales 27

1.5. Patrones de diseño..... 27

1.6. Pruebas 28

Conclusiones parciales……...29

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN……..30

2. Introducción 30

2.1. Descripción de la solución 30

2.2. Requisitos..... 30

2.2.1. Técnicas de identificación de requisitos..... 31

2.2.2. Especificación de requisitos de software 31

2.3. Historias de usuarios 34

2.4. Planificación de iteraciones 35

2.5. Diseño 37

2.5.1. Diseño arquitectónico..... 37

2.5.2. Modelado del diseño 38
2.6. Patrones del diseño de software 39
Conclusiones parciales..... 41
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS..... 42
3. Introducción 42
3.1. Implementación 42
3.1.1. Estructura y precedencia de operadores 42
3.1.2. Orden y restricciones establecidas para la creación de consultas 43
3.1.3. Algoritmo de transformación de consultas SQL y DQL a partir del modelo 45
3.1.4. Generación de archivos de datos 47
3.1.5. Estándares de codificación 47
3.2. Pruebas 48
3.2.1. Pruebas unitarias 48
3.2.2. Pruebas de aceptación 50
3.2.3. Validación del objetivo de la investigación..... 52
Conclusiones parciales..... 54
CONCLUSIONES GENERALES..... 55
RECOMENDACIONES..... 56
REFERENCIAS BIBLIOGRÁFICAS..... 57

ÍNDICE DE FIGURAS

Figura 1: Técnica Estrella de Bhoem y Turner, para la selección del enfoque metodológico de desarrollo de software 22

Figura 2: Vista de arquitectónica del sistema 37

Figura 3: Patrón Controlador 40

Figura 4: Patrón Creador 41

Figura 5: Distribución de los operadores 43

Figura 6: Utilización del estilo UpperCamel Case en declaración de clases 47

Figura 7: Utilización del estilo LowerCamel Case en los nombres de variables 48

Figura 8: Utilización del estilo LowerCamel Case en la declaración de métodos 48

Figura 9: Resultados de las pruebas unitarias 50

Figura 10: Resultados de las pruebas de aceptación 52

Figura 11: Resultados de la comparación de los tiempos empleado en la realización de las consultas 53

Figura 12: Resultados de la comparación de los tiempos empleados para la elaboración de archivos de datos 54

ÍNDICE DE TABLAS

Tabla 1: Requisitos funcionales obtenidos para el sistema 32

Tabla 2: Historia de usuario Guardar código DQL 34

Tabla 3: Resultado de la Planificación 36

Tabla 4:Tarjeta CRC clase Consulta.java 38

Tabla 5: Tarjeta CRC clase Fixture.java 39

Tabla 6: Restricciones por cláusulas 43

Tabla 7: Distribución de operadores 44

Tabla 8: Caso de prueba de aceptación de la HU Gestionar condición where 50

INTRODUCCIÓN

El uso de las tecnologías de la información y las comunicaciones se ha convertido en el componente central de toda empresa o negocio que busque mejorar su rendimiento en el menor tiempo posible. Este objetivo puede ser cumplido mediante herramientas y aplicaciones de software que realicen un procesamiento eficaz de la información que en ellas se maneja, mediante sistemas de base de datos encargados de persistirla, para un posterior uso. Una Base de Datos (BD) se define como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular (Sánchez, 2004).

Uno de los objetivos fundamentales de un Sistema Gestor de Bases de Datos (SGBD) es proporcionar a los usuarios una visión abstracta de los datos, es decir, el usuario va a utilizar esos datos, pero no tendrá idea de cómo estarán almacenados físicamente. Los modelos de datos son el instrumento principal para ofrecer esta abstracción. Son utilizados para la representación y el tratamiento de los problemas (Sánchez, 2004). Entre los diferentes modelos de base de datos que existen se encuentran los modelos conceptuales, los lógicos y los físicos (Sánchez, 2004)**.** Siendo las BD relacionales un ejemplo representativo del modelo físico, las cuales son las más comunes y extendidas, ya que permiten modelar diversas situaciones mediante las relaciones entre tablas o entidades, objetivo que precede su creación.

En la actualidad existe un número creciente de marcos de trabajo (*framework*) para una amplia diversidad de lenguajes y/o tecnologías, algunos de ellos facilitan el trabajo con las bases de datos en el desarrollo de aplicaciones informáticas. La utilización de estos marcos de trabajo ha reportado beneficios considerables en cuanto a reducción de tiempos y errores durante la fase de implementación en el proceso de desarrollo (Khan, 2005). Estos emplean la técnica de programación denominada mapeo de objetos-relacional (ORM, por sus siglas en inglés), la cual consiste en *"la persistencia automatizada y transparente de las tablas en una Base de Datos relacional, usando metadatos*[*]*que definen el mapeo entre los objetos y la Base de Datos"* (Bauer, y otros, 2004)*.*

[*] Metadatos: datos que describen otros datos.

En la actualidad, es común la utilización de marcos de trabajo ORM para facilitar el acceso a los datos de los programadores. En el ámbito de las aplicaciones web uno de los ORM que más gana adeptos para su uso dentro del desarrollo con el lenguaje de programación PHP es el Doctrine, por poseer una documentación que ha estado en constante crecimiento, así como una comunidad activa y un bajo nivel de configuración para iniciar un proyecto. Mediante la singular notación que provee el mismo, se puede llegar a describir con precisión las entidades, atributos y las relaciones que se utilizan para almacenar y tratar la información. Además, este ORM, a través de archivos de datos (*data fixture*, en inglés*)*, brinda al usuario la posibilidad de cargar datos que resultan necesarios para el funcionamiento de la solución que se desarrolla o para las pruebas en tiempo de desarrollo.

La Universidad de las Ciencias Informáticas (UCI), es uno de los principales productores de software del país y parte activa del proceso de informatización que impulsa el gobierno cubano. Está concebida en facultades, las cuales cuentan con centros de desarrollo, donde se llevan a cabo procesos complejos con avanzadas herramientas y técnicas. Dentro de ellos se encuentra el Centro de Gobierno Electrónico (CEGEL), el mismo tiene como objetivo satisfacer necesidades de entidades y organizaciones mediante el desarrollo de productos, servicios y soluciones informáticas para la gestión de las áreas de gobierno (CEGEL). En aras de alcanzar tales fines, se emplean un conjunto de herramientas y tecnologías dentro de las que se encuentran Doctrine para realizar la persistencia de la información en bases de datos relacionales.

El proceso de mapeo de grandes volúmenes de clases en negocios con alto nivel de complejidad dio lugar, dentro de CEGEL, al desarrollo de la herramienta ORMAPING para el trabajo con Doctrine. Esta herramienta permite a los usuarios mantener sincronizadas las entidades y sus correspondientes tablas en la BD, así como diseñar el modelo de datos a utilizarse durante la implementación de las distintas soluciones, pero carece de funcionalidades para trabajar el acceso a datos.

Los arquitectos de datos de CEGEL en el proceso de gestión de datos se encuentran con las siguientes situaciones:

- Para la ejecución de consulta a los datos, los desarrolladores necesitan conocimientos de lenguajes como SQL (*StructuredQueryLanguage*) o DQL (*Doctrine QueryLanguage*)

y estos son directamente proporcionales a la complejidad de la consulta que se necesita, lo que provoca que el desarrollo de las mismas implique más tiempo y/o personal cualificado.

- Las pruebas de las consultas, haciendo uso del *framework*, necesitan conocimiento de la arquitectura empleada o de comandos específicos en el terminal, trayendo consigo los mismos inconvenientes descritos en el caso anterior.
- Cuando los datos necesarios para el funcionamiento de las soluciones que emplea el ORM Doctrine se encuentran ya en otra base de datos u archivo, no existe un mecanismo automatizado para su inclusión en los archivos de datos; por lo que se hace codificando en la herramienta en la que se esté desarrollando, acarreando un costo de tiempo de desarrollo por este concepto.
- El desarrollo de los archivos de datos se realiza codificando directamente en el entorno de desarrollo integrado (IDE, por sus siglas en inglés) lo que aumenta la posibilidad de cometer errores y el tiempo empleado en el desarrollo.

En correspondencia con el contexto descrito, en la presente investigación se identificó el siguiente **problema a resolver:** ¿Cómo contribuir a la disminución del tiempo empleado en la gestión de datos de los proyectos productivos de CEGEL?

Este problema se enmarca en el **objeto de estudio:** proceso de mantenimiento de software, identificándose como **campo de acción:** mantenimiento de software en aplicaciones de escritorio.

Para resolver el problema identificado se definió el siguiente **objetivo general:** desarrollar las funcionalidades que permitan la realización visual de consultas y archivos de datos, para disminuir el tiempo de realización de las tareas de los arquitectos de datos de CEGEL.

Objetivos específicos:

- Elaborar el marco teórico en función de los conceptos, términos y herramientas necesarias para abordar adecuadamente la solución del problema.
- Desarrollar la propuesta de software que permita dar solución al objetivo general de la investigación.
- Validar la solución propuesta mediante su aplicación directa en un entorno controlado.

Para dar cumplimiento a los objetivos anteriormente planteado se definen las siguientes **tareas de investigación:**

1. Análisis de los principales conceptos, términos y herramientas necesarias para abordar adecuadamente la solución del problema.
2. Selección de las herramientas y la metodología de desarrollo a utilizar en el desarrollo de la solución.
3. Especificación de los requisitos funcionales y no funcionales a tener en cuenta para el desarrollo de la solución.
4. Implementación de la solución a partir de las herramientas seleccionadas y los requisitos identificados.
5. Aplicación de pruebas a la solución que permitan validar la solución desarrollada.
6. Ejecución de las pruebas piloto para comprobar el cumplimiento del objetivo general de la investigación.

Para guiar la investigación se plantea la siguiente **idea a defender:** El desarrollo de funcionalidades para el desarrollo visual de consultas y archivos de datos, permitirá disminuir el tiempo de realización de las tareas de los arquitectos de datos de CEGEL.

Para la realización de las tareas de investigación se utilizaron los métodos científicos siguientes:

Métodos teóricos:

- **Histórico – lógico:** para determinar los antecedentes y tendencias en el proceso de realización de consultas y archivos de datos a través del marco de trabajo Doctrine.
- **Analítico – sintético:** para descomponer el problema de la investigación en elementos por separado y profundizar en el estudio de cada uno de ellos y luego sintetizarlos en la solución de la propuesta.
- **Inductivo – deductivo:** este método se utiliza para concluir nuevos conocimientos y predicciones referentes al proceso de desarrollo de consultas y archivos de datos con el ORM Doctrine en CEGEL, que posteriormente son sometidos a verificaciones empíricas.

- **Modelación:** posibilitó la obtención de los principales artefactos de cada una de las fases de desarrollo del sistema.

Métodos empíricos:

- **Entrevista:** se empleó con el objetivo de conocer la estructura del proceso de realización de consultas y archivos de datos en el marco de trabajo Doctrine empleado en CEGEL y los problemas asociados al mismo.
- **Medición:** con el objetivo de obtener información numérica acerca del proceso de mapeo de objetos-relacional del marco de trabajo Doctrine en CEGEL. Además, se utiliza para evaluar el comportamiento del software diseñado y su pertinencia con los requisitos establecidos.

La tesis consta de introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos. El documento está estructurado de la siguiente forma:

En el ***Capítulo 1. Fundamentación teórica****,* se describen los términos más importantes asociados al proceso de mapeo objetos relacionales, así como elementos esenciales de las aplicaciones existentes para realizar dicho proceso. Además, se realiza el estudio de los conceptos fundamentales relacionados con el tema en cuestión. Al final, se define la metodología para guiar el desarrollo de la investigación, las tecnologías y las herramientas que serán utilizadas para la construcción de la solución.

En el ***Capítulo 2. Análisis y diseño de la propuesta de solución***, se abarcan los principales elementos del desarrollo del sistema propuesto, la especificación de sus funcionalidades, prefijación de la ejecución de las mismas, la arquitectura y modelación del sistema de acuerdo a lo propuesto por la metodología elegida.

En el ***Capítulo 3. Implementación y pruebas****,* se describe la solución desde el punto de vista técnico, se evidencia el empleo de patrones y estándares de codificación. Además, se define la estrategia de pruebas a partir de la metodología de desarrollo elegida, con el objetivo de garantizar la validación de la solución propuesta. Se analizan los niveles de pruebas ejecutados y los resultados alcanzados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1. Introducción

En el presente capítulo se abordan conceptos, que por su relación con la temática tratada es importante definir. Además, se describen las herramientas, lenguajes y tecnologías de desarrollo seleccionadas para la implementación de la solución que se propone, así como los patrones y estrategias de pruebas posibles a emplear dando como resultado la actualización del estado del arte de la investigación.

1.1. Conceptos fundamentales asociados al dominio del problema

1.1.1. Mantenimiento de Software

En ingeniería del software, el mantenimiento es la modificación de un producto de software después de la entrega, para corregir errores, mejorar el rendimiento u otros atributos. Y a decir de Barreiro es una de las actividades más comunes dentro del proceso de desarrollo de software.

El mantenimiento se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software, y a cambios debido a las mejoras producidas por los requisitos cambiantes del cliente (Pressman, 2003).

Tipos de mantenimiento de software (Berriero, 2011)**:**

- **Mantenimiento preventivo:** consiste en la revisión constante del producto con el fin de detectar nuevos focos de problemas que puedan seguir surgiendo en el futuro.
- **Mantenimiento correctivo:** corrige los defectos encontrados en el software y que originan un comportamiento distinto al deseado.
- **Mantenimiento adaptativo:** si se quiere cambiar el entorno de uso de la aplicación, puede ser indispensable modificarla para mantener su plena funcionalidad en estas nuevas condiciones.
- **Mantenimiento evolutivo:** es un caso especial donde la adaptación resulta prácticamente obligatoria, ya que de lo contrario el programa quedaría obsoleto con el paso del tiempo.

- **Mantenimiento perfectivo:** por distintas razones el usuario puede solicitar el agregado de nuevas funcionalidades o características no contempladas al momento de la implementación del software. Este tipo de mantenimiento adapta la aplicación a este requerimiento.

Actividades generales para el mantenimiento de software (Sommerville, 2005)**:**

- **Comprensión del software y de los cambios a realizar**: para poder modificar un programa, los programadores necesitan conocer su funcionalidad y objetivos, su estructura interna y los requisitos de operación. De no ser así, se corre un gran riesgo de introducir nuevos defectos que en el futuro supondrán un coste de mantenimiento adicional.
- **Modificación del software**: para incorporar los cambios necesarios se deben crear y modificar las estructuras de datos, la lógica de los procesos, las interfaces y la documentación. Los programadores deben conocer lo mejor posible las repercusiones que tienen en el sistema los cambios que están realizando, con el fin de evitar al máximo posible los efectos secundarios.
- **Realización de pruebas**: para validar los cambios se deben realizar pruebas selectivas que nos permitan comprobar la corrección del software. Esta actividad es necesaria siempre, ya que incluso un cambio muy pequeño no verificado puede producir defectos en el software que reduzcan su calidad y fiabilidad.

Una vez descritos algunos de los aspectos fundamentales de los tipos de mantenimiento de software se realiza un análisis de la herramienta ORMaping con el objetivo de identificar algunos puntos de mejora y características del mismo que puedan indicar el tipo de mantenimiento a emplear.

1.1.2. Análisis de ORMaping

ORMaping, es una aplicación de escritorio de código abierto, multiplataforma, desarrollada bajo el lenguaje java en su versión 1.7, para el trabajo con el ORM Doctrine, la cual ofrece funcionalidades de mapeo que apoyan el trabajo de los equipos de desarrollo de CEGEL. Fue concebido a los arquitectos de datos del centro, que necesiten realizar el mapeo de grandes volúmenes de clases en negocios complejos mediante Doctrine, un ambiente de desarrollo

amigable y fácil de usar, que cubra sus necesidades respecto al tiempo de realización de sus tareas. La herramienta proporciona la posibilidad de generar la misma lógica de negocio que genera el marco de trabajo Doctrine (Ricart, y otros, 2015).

Funcionamiento

Si es la primera vez que se ejecuta, la aplicación brinda una interfaz donde el usuario puede crear el diagrama de clases persistentes, de forma manual o importarlo desde las entidades con extensión (.php), donde se encuentra el mapeo relacional de objetos. Además, se puede configurar la conexión con el gestor postgresql, a partir de ello se realizan básicamente tres operaciones: probar la conexión, de acuerdo a los datos introducidos cuando se configura la base de datos, luego se puede generar una nueva base de datos o crear las tablas dentro de esta. Para cada objeto del diagrama de clases persistentes, el sistema genera las clases php, de igual forma crea un archivo script (.sql), el cual puede ser ejecutado posteriormente por el SGBD (Ricart, y otros, 2015).

Este proyecto después puede ser guardado con una extensión propia de la aplicación .map y puede ser reeditado posteriormente. El software tiene una funcionalidad distintiva y es que permite sincronizar las entidades y la base de datos, cuando es necesario realizar alguna modificación en el proyecto se abre el archivo donde se guarda el diagrama o se pueden importar las entidades para realizar los cambios. ORMaping permite que todo el trabajo se realice a través de una interfaz visual fácil de manipular por los arquitectos de datos.

Ventajas de ORMaping:

- Permite a los usuarios mantener sincronizadas las entidades y sus correspondientes tablas en la base de datos, así como diseñar el modelo de datos a utilizarse durante la implementación de las distintas soluciones.
- El empleo de la herramienta ha reportado beneficios considerables en cuanto a la reducción del tiempo empleado para el mapeo de grandes volúmenes de datos dentro de CEGEL.

Limitaciones de ORMaping:

- **Funcionales:** carece de soluciones para manejar acceso a datos y el desarrollo de consultas.

- **Soporte:** solo permite la conexión con postgresql como sistema gestor de base de datos.

Se tuvo en cuenta el análisis de la herramienta y el alto valor que representa dentro de los proyectos productivos de CEGEL y el correcto funcionamiento de la misma, además de los beneficios que reporta, resultando que no se han detectado nuevos focos de problemas o imperfecciones, además los usuarios están familiarizados con la interfaz por lo que se considera que no se debe cambiar el entorno de trabajo. Además, no se han detectado funcionamientos anómalos distintos a los deseados, sino que, a pesar de su novedad, se desea que el sistema incorpore a sus prestaciones nuevas funcionalidades y características explotables por los usuarios finales. Se decide aplicar entonces un mantenimiento de tipo perfectivo, con el fin de incorporar nuevas funcionalidades que faciliten el trabajo de los arquitectos de datos de CEGEL.

1.1.3. Doctrine

Doctrine es un ORM escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un SGBD. La principal tarea de los ORM para PHP es la traducción transparente entre objetos y las filas relacionales de la base de datos (Carrero , 2013).

Una de las principales características de Doctrine es su lenguaje de consulta estructurado llamado Lenguaje de Consulta de Doctrine (DQL), inspirado en el Lenguaje de Consulta de Hibernate (HQL, por sus siglas en inglés). Además, DQL difiere de SQL en que abstrae considerablemente la asignación entre las filas de la base de datos y objetos, permitiendo a los desarrolladores escribir poderosas consultas de una manera sencilla y flexible. Soporta las operaciones de Crear, Leer, Actualizar y Borrar (CRUD - *Create, Read, Update and Delete*) habituales, desde la creación de nuevos registros a la actualización de los antiguos. Crea manual y automáticamente el modelo de base de datos a implementar. Soporta varios motores de bases de datos como MySQL y PostgreSQL (Carrero , 2013).

Características principales de Doctrine (Pacheco, 2011):

- Generación automática del modelo: crear el conjunto de clases que representa el modelo de la aplicación, luego, estas clases serán vinculadas al esquema de la base de datos de forma automática con un motor ORM.
- Posibilidad de trabajar con anotaciones: mecanismo muy utilizado en el lenguaje de programación Java, las aplicaciones del *framework* Symfony2[†] pueden hacer uso de ellas gracias a una librería desarrollada por el proyecto Doctrine 2.
- Relaciones entre entidades: una vez que se ha definido el modelo (o se ha creado de forma automática) con las tablas y sus relaciones, resulta fácil acceder y moverse por entidades relacionadas.

Las versiones recientes del marco de trabajo Doctrine están registradas a partir de Doctrine 2, el cual marca un nuevo enfoque ORM. El mismo trae consigo varias ventajas que se especifican a continuación (Carrero , 2013).

- Sus objetos persistentes (llamados entidades en Doctrine 2) no están obligados a extender más de una clase base abstracta. Doctrine 2 permite el uso de simples objetos de PHP.
- Presenta una mejor arquitectura y potentes algoritmos que logran realizar un funcionamiento más rápido en comparación con la versión anterior.

1.1.4. Datos de prueba

Se utilizan para cargar un conjunto de datos a través de sus modelos para poblarla base de datos. Los mismos se utilizan a menudo al lado del otro con algún tipo de unidad de paquete de evaluación funcional (Team, 2016).

1.2. Metodología, lenguajes y herramientas de desarrollo

En todos los proyectos de desarrollo de software un paso fundamental es la elección de la metodología, lenguajes y herramientas de desarrollo, a continuación, se describen los principales argumentos considerados para la ejecución del presente trabajo de diploma.

[†] Conjunto de componentes PHP que conforman un marco de trabajo (*framework*) para desarrollar aplicaciones web.

1.2.1. Metodología de desarrollo

La metodología constituye la columna vertebral de todo el proceso de desarrollo de software. Se define como metodología a: “una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo” (Letelier, 2003).

Teniendo en cuenta las tareas generales para la realización del mantenimiento de software y los posibles problemas que surgen durante el desarrollo de las mismas, así como la posibilidad de contar con un conjunto de artefactos e información, referente a la herramienta a la cual se pretende aplicar mantenimiento y dadas las características del equipo de desarrollado y el cliente, formando parte de este, además de la prontitud con la que se desea desarrollar la solución, se elige la metodología de desarrollo de software de enfoque ágil Programación Extrema (XP por sus siglas en inglés); para apoyar la decisión se aplica la técnica Estrella Bhoem y Turner enmarcada para la selección del enfoque.

La técnica antes mencionada plantea 5 criterios fundamentales mediante los que se estará valorando el proyecto; estos son: tamaño del equipo, criticidad del producto, dinamismo de los cambios, cultura del equipo y personal con que se cuenta. Cada uno de esos criterios tiene elementos que lo discriminan y por tanto se tienen en cuenta a la horade seleccionar uno u otro enfoque (Gabardini, 2004). Para la selección del valor que se ubicará en cada eje (uno para cada criterio) de la estrella se debe tener en cuenta el comportamiento de estos criterios en el proyecto.

En lo sucesivo se describe cada uno para el presente caso de estudio:

- Tamaño: el equipo de desarrollo está formado por 1 programador que debe dar respuesta a 33 requerimientos. Estos requerimientos por las características del proyecto y teniendo en cuenta que el cliente forma parte del equipo, están sujetos a posibles cambios durante la etapa de desarrollo.

- Criticidad: el sistema cuenta con una baja criticidad, orientado a brindar utilidades dentro del entorno social o al usuario final para el que está destinado, lo que no constituye un riesgo para la vida o pérdidas económicas considerables.
- Dinamismo: de los requerimientos planteados durante el proceso de ingeniería de requisitos se detectó que los requisitos son cambiantes, basados fundamentalmente en el desarrollo de las TIC y las nuevas necesidades organizacionales. Por lo que se prevé que la lógica de negocio o el ambiente de trabajo para el cual está destinado puede variar con facilidad.
- Personal: el personal del equipo es graduado de nivel medio superior, con conocimientos adquiridos durante los 5 años de estudios universitarios de la especialidad, aún sin referentes prácticos en el área de desarrollo de software. Por otra parte, muestra un alto compromiso y responsabilidad ante la tarea asignada.
- Cultura: considerada alta, fundamentalmente porque solo posee un miembro el cual tiene bien definida sus tareas y responsabilidades, de modo que no se necesita una estructura jerárquica dentro del proyecto.

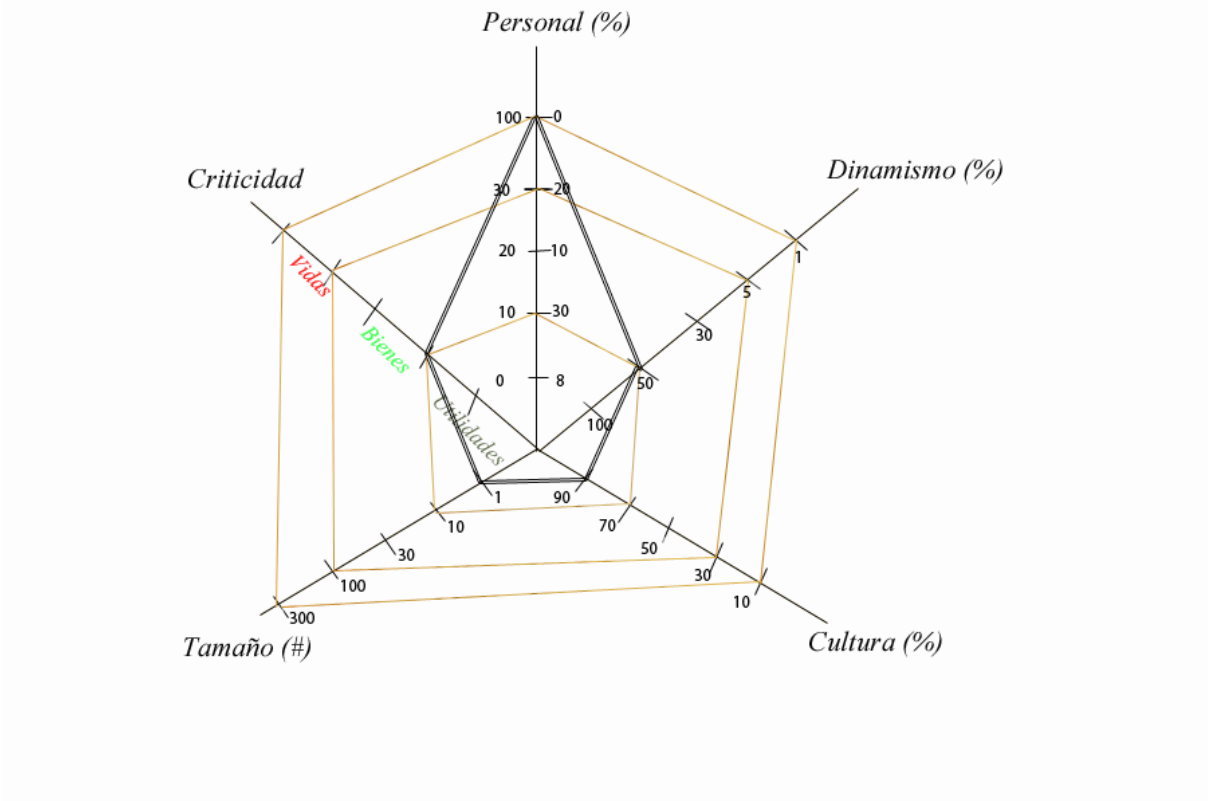

Figura 1: Técnica Estrella de Bhoem y Turner, para la selección del enfoque metodológico de desarrollo de software

Como se puede apreciar, al realizar el análisis, basado en todas las características del proyecto, la técnica arroja en el 80% de sus puntos un enfoque ágil por lo que finalmente se aprueba XP como metodología de desarrollo para esta propuesta pues de esta forma se potencia la reutilización de la información y artefactos ya existentes de la versión anterior de la aplicación.

Esta metodología potencia las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen ambiente de trabajo.

La misma propone las siguientes fases (Beck, 2000):

- **Planificación:** en esta fase se elabora el plan de iteraciones a ejecutar durante el desarrollo del proyecto a partir de la prioridad de los requisitos especificados en las historias de usuario y el esfuerzo necesario para su implementación.
- **Diseño:** es el proceso para definir la arquitectura, los componentes, las interfaces y se define el diseño del sistema a partir de la modelación de sus clases en tarjetas de Clase Responsabilidad Colaboración (CRC). Una correcta especificación de los nombres de métodos y clases, ayuda a comprender el diseño y facilita sus posteriores ampliaciones y la reutilización del código.
- **Implementación:** la fase de implementación 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.
- **Pruebas:** uno de los pilares de la metodología XP es la ejecución constante de pruebas para comprobar el funcionamiento de los códigos implementados. Se definen dos niveles de pruebas: pruebas de unidad y pruebas de aceptación. Las pruebas de unidad están enfocadas a validar la solución desde el punto de vista técnico, a evaluar el código generado, en tanto las pruebas de aceptación las realiza el cliente para verificar que las funcionalidades implementadas cumplen con los requisitos acordados. Un punto importante de las pruebas de unidad es su automatización, además se deben diseñar antes que las funcionalidades a probar, garantizando la independencia del código a evaluar.

1.2.2. Lenguaje de modelado y herramienta case

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de *software*. Prescribe un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, y describir la semántica esencial de lo que estos diagramas y símbolos significan. UML se puede usar para modelar distintos tipos de sistemas: sistemas de *software*, sistemas de hardware y organizaciones del mundo real (Feriñas, 2013).

Visual Paradigm es una herramienta de Ingeniería de Software Asistida por Ordenador (CASE), desarrollada por la compañía Visual Paradigm International, que utiliza UML como lenguaje de modelado. Permite soportar el ciclo de vida completo del *software*: análisis, diseño, implementación y despliegue. La captura de requisitos, el dibujo de diagramas UML, la realización de ingeniería inversa y generación de código C++ y Java. Puede ser integrado con varias herramientas como son: Eclipse/IBM, Builder, Net Beans IDE, Oracle JDeveloper, BEA Weblogic. Está disponible en varias ediciones, cada una destinada a necesidades específicas: Enterprise, Professional, Community, Standard, Modeler y Personal.

La versión empleada en la solución es el Visual Paradigm 5.0, lanzada el 16 de agosto del 2010.

1.2.3. Lenguaje de programación y entorno de desarrollo integrado

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; permitiendo al programador comunicarse con los dispositivos hardware y software existentes (Tecnología, 2014).

Para el desarrollo en cuestión se selecciona **Java** como lenguaje de programación por ser este el lenguaje sobre el que está implementado el software, al cual se le aplicará el mantenimiento y la capacidad plena de este lenguaje, para satisfacer las necesidades de implementación.

Java

Es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems en la década del noventa. La intención era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel (Fernandez, 2004).

Uno de sus pilares es la posibilidad de ejecutar un mismo programa en diversos sistemas operativos (independiente de la plataforma). Además, todo programa desarrollado en Java ha de compilarse y el código que se genera es interpretado por una máquina virtual, de este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que son dependientes de la plataforma. Su diseño presenta como características ser robusto, seguro, portable, independiente a la arquitectura, dinámico e interpretado. En la implementación de la aplicación es utilizado Java por ser el lenguaje que permite tener acceso a las funcionalidades administradas por la Interfaz de Programación de Aplicaciones (API, según sus siglas en inglés) (Fernandez, 2004).

Debido a que el entorno de ejecución objetivo de la herramienta será los laboratorios de producción de CEGEL y la versión de la máquina virtual de java allí instalada es al menos superior a la 1.7 se tomará dicha versión para el desarrollo de la aplicación teniendo en cuenta que en caso de existir versiones superiores se asegure su compatibilidad.

1.2.4. Entorno de desarrollo integrado (IDE)

Un IDE es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien poder utilizarse para varios. Puede denominarse como un entorno de programación que ha sido tratado como un programa aplicación. Esto significa que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (Editorbfb, 2011).

Netbeans

Es un IDE gratuito y de código abierto. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web o para dispositivos móviles. Da soporte

a las siguientes tecnologías: Java, PHP, C/C++, HTML. Además, puede instalarse en varios sistemas operativos: Windows, Linux, Mac OS (Netbeans.org, 2012).

Características principales que permitieron emplearlo en la solución (NetBeans, 2014):

- Brinda soporte a casi todas las novedades en el lenguaje Java. Cualquier vista preliminar del lenguaje es rápidamente soportada por Netbeans.
- Incorpora asistentes para la creación y configuración de distintos proyectos, incluida la elección de algunos marcos de trabajo.
- Posee un editor de código robusto, multilenguaje, con el habitual coloreado y sugerencias de código, acceso a clases a través de enlaces en el código, control de versiones, ubicación de la clase actual, comprobaciones sintácticas, semánticas y plantillas de código.
- Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario.
- Contiene herramientas para depurado de errores: el depurador que incluye el IDE es bastante útil para encontrar fallos en el código.

1.3. Sistema gestor de bases de datos (SGBD)

Un SGBD es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos (Sánchez, 2004).

PostgreSQL 9.2

Es un SGBD libre, publicado bajo la licencia BSD y liberado el 10/09/2012. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una sola empresa, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Mediante un sistema denominado Acceso Concurrente Multiversión (MVCC), PostgreSQL permite que mientras un proceso escribe en

una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Esta estrategia es superior al uso de bloqueos por tabla o por filas, común en otros SGBD, eliminando la necesidad de uso de bloqueos explícitos (PostgreSQL, 2003).

1.4. Bibliotecas adicionales

Para el desarrollo de la solución se hará uso de las siguientes bibliotecas de código de terceros, libres y especializadas en funcionalidades específicas:

Marco de trabajo jUnit 4.10: el objetivo del uso de jUnit con Netbeans IDE es escribir y ejecutar pruebas fácilmente repetibles. jUnit es una instancia de la arquitectura xUnit para marcos de trabajo encargados de realizar pruebas unitarias, fue desarrollado por el autor de la metodología XP y está orientado a la realización de pruebas de unidad en el lenguaje Java por parte del programador (Ing. Oré , 2009).

JGraphX: es una biblioteca de visualización de diagramas de Java Swing licenciada bajo la licencia NewBSD (*New Berkeley Software Distribution*). Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre. Originalmente la librería fue nombrada JGraph a secas hasta la versión 1.5. En la versión 1.6 se decidió cambiar el nombre a JGraphX para reflejar que el código base de la API fue reescrito desde cero (jGraphx, 2006).

JGraphX proporciona funcionalidades para la visualización e interacción con grafos (no gráficos). Algunos ejemplos de aplicaciones que pueden ser escritos con esta librería son editores de flujos de trabajo, cuadros de organizaciones, herramientas de modelado de procesos de negocios, visualizador de circuitos electrónicos, visualizador de redes de telecomunicaciones, entre otros (jGraphx, 2006).

1.5. Patrones de diseño

Los patrones de diseño son descripciones estructurales y funcionales de cómo resolver, de forma concreta un determinado problema mediante orientación a objetos. Permiten que el diseño sea más flexible, elegante y reutilizable (Montenegro Jiménez, y otros, Agosto/Diciembre 2012).

- Patrones Generales de Software para la Asignación de Responsabilidades *General (Responsibility Assignment Software Patterns*, por sus siglas en inglés): describen un conjunto de principios básicos de la asignación de responsabilidades a objetos. Algunos de los patrones que conforman este grupo son: experto, creador, bajo acoplamiento, alta cohesión y controlador (Larman, 2004).
- Patrones Banda de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (GOF, por sus siglas en inglés): según su propósito, se clasifican en tres categorías: creacionales, estructurales y de comportamiento. Los patrones creacionales se encargan de la creación de instancias de los objetos. Los estructurales separan la interfaz de la implementación, además, plantean las relaciones entre clases, las combinan y forman estructuras mayores. Los patrones de comportamiento describen la comunicación entre objetos y clases (Guerrero, y otros, 2013).

1.6. Pruebas

La metodología XP propone el empleo de 2 niveles de pruebas para la validación de la solución: pruebas de unidad y de aceptación, las cuales se realizan durante todo el ciclo de desarrollo de software (Rodríguez Tello, 2012).

Las **pruebas de unidad** son una forma de probar el correcto funcionamiento de un módulo o una parte del sistema. Con el fin de asegurar que todos los módulos cumplan con lo requerido cada uno por separado y evitar así errores futuros en el momento de la integración de todas sus partes. La idea es escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto (Malfará, 2006).

Las **pruebas de aceptación** son pruebas definidas por el cliente para cada historia de usuario, con el objetivo de asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. En efecto marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención (Malfará, 2006).

Conclusiones parciales

Con el desarrollo del presente capítulo queda constituido el marco teórico referencial de la investigación. Luego de analizadas las necesidades del cliente y los conceptos de mantenimiento estudiados, así como las principales características de la herramienta ORMaping, se determinó realizar un mantenimiento de tipo perfectivo ya que no se realizarán cambios críticos en el software, solo ajustes necesarios para la incorporación de las nuevas funcionalidades. El lenguaje a emplear para el desarrollo de la solución será java en su versión 1.7 mientras que Netbeans en su versión 8 dará soporte como entorno de desarrollo integrado. Las bibliotecas JUnit en su versión 4.1 y jGraphx en su versión 3.5 complementarán el desarrollo ofreciendo componentes o funciones que facilitan o mejoran la eficacia del trabajo. La eficiencia y calidad serán validadas con la realización de pruebas unitarias y el trabajo con interfaces graficas respectivamente. Todo el proceso antes descrito se regirá por la metodología ágil XP para fundamentar y establecer un mecanismo de trabajo estándar comprensible por otros equipos de desarrollo para lo cual también se describen los patrones para implementación y diseño de la solución, así como la estrategia de pruebas a emplear.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

2. Introducción

En el capítulo se realiza la descripción de las principales definiciones asociadas al dominio del problema. También se especifican los requisitos funcionales y no funcionales que tienen lugar en la implementación. Además, se describen los artefactos generados por esta fase teniendo en cuenta la metodología seleccionada. Igualmente, se especifican los patrones de diseño y la arquitectura utilizada en la concepción de la solución.

2.1. Descripción de la solución

La solución propuesta consiste en una nueva versión de la ORM MAPPING, con el desarrollo de la misma se pretende dar solución a los problemas que tienen los equipos de desarrollo, a la hora de realizar consultas y generar datos de prueba, brindándole la posibilidad de construir las consultas de forma visual sin necesidad de conocer los lenguajes empleados para su realización así mismo el sistema permite la generación de datos de prueba a partir de una base de datos teniendo en cuenta la relaciones existentes entre las tablas que componen dicha base de datos y posteriormente exportarlos para ser utilizados en el proyecto en cuestión.

2.2. Requisitos

La ingeniería de requisitos constituye un elemento fundamental en todo proceso de desarrollo de software que tenga como finalidad la obtención de sistemas informáticos que se ajusten a las necesidades reales de los clientes, dichos requisitos tienen como objetivo establecer los servicios que el sistema deberá proveer y las restricciones bajo las cuales deberá ser desarrollado. Trata de los principios, métodos, técnicas y herramientas que permiten descubrir, documentar y mantener los requisitos para sistemas basados en computadora, de forma sistemática y repetible (Guibert Estrada, 2011). Lo anteriormente expuesto se puede interpretar como que el propósito de los requisitos es hacer que los sistemas alcancen un estado óptimo antes de alcanzar la fase de diseño en el proyecto. A continuación, se presentan las actividades desarrolladas en esta disciplina, así como los productos de trabajo elaborados.

2.2.1. Técnicas de identificación de requisitos

Para realizar una adecuada identificación de requisitos existen diversas técnicas que guían al analista en el proceso de comunicación con el cliente y el equipo de desarrollo como las que a continuación se presentan.

Entrevista: esta técnica se empleó con el fin de lograr un entendimiento del problema de forma clara y transparente permitiendo situar al cliente en un punto en el cual el mismo obtuviera opinión precisa de cuáles son los problemas que presenta y de cómo solucionarlos desde el punto de vista del desarrollador.

Tormenta de ideas: esta técnica se utilizó para identificar los requisitos iniciales con los que debería contar la aplicación. Para aplicarla se realizaron varias reuniones entre el cliente y el equipo de desarrollo donde ambas partes brindaban sus ideas en cuanto a la propuesta de solución.

2.2.2. Especificación de requisitos de software

La especificación de requisitos de software tiene como objetivo principal servir como **medio de comunicación** entre clientes, usuarios y equipo de desarrollo. Contiene un conjunto de información que ayuda a los desarrolladores de software a analizar y entender todos los requisitos que el cliente desea así como los requisitos que debe cumplir el sistema a desarrollar para satisfacer dichas necesidades (Martínez Guerrero).

Requisitos funcionales

Los requerimientos o requisitos funcionales (RF) de un software son la descripción de los servicios proporcionados por el mismo y sus restricciones operativas. Reflejan las necesidades de los clientes resueltas mediante un sistema informático (Sommerville, 2005).A continuación, se relaciona el listado de requisitos convenidos con el cliente:

Tabla 1: Requisitos funcionales obtenidos para el sistema

Agrupación de requisitos	RF	Nombre
Gestionar datos de prueba	RF. 1	Crear dato de prueba
	RF. 2	Modificar dato de prueba
	RF. 3	Listar dato de prueba
	RF. 4	Eliminar dato de prueba
	RF. 5	Exportar dato de prueba
Requerimientos sin agrupación	RF. 6	Construir consulta
	RF. 7	Visualizar código DQL
	RF. 8	Visualizar código SQL
	RF. 9	Visualizar resultado de la consulta
	RF. 10	Guardar código DQL y SQL
Gestionar entidades	RF. 11	Adicionar entidades
	RF. 12	Eliminar entidades
	RF. 13	Listar entidades
Gestionar condición Select	RF. 14	Adicionar condición select
	RF. 15	Eliminar condición select
	RF. 16	Listar condiciones select
Gestionar unión entre entidades	RF. 17	Adicionar unión
	RF. 18	Eliminar unión
	RF. 19	Listar uniones
Gestionar condición Where	RF. 20	Adicionar condición where
	RF. 21	Modificar condición where
	RF. 22	Eliminar condición where
	RF. 23	Listar condición where
Gestionar condición Having	RF. 24	Adicionar condición Having
	RF. 25	Modificar condición Having
	RF. 26	Eliminar condición Having
	RF. 27	Listar condición Having
Gestionar condición GroupBy	RF. 28	Adicionar condición GroupBy
	RF. 29	Eliminar condición GroupBy
	RF. 30	Listar condición GroupBy
Gestionar condición OrderBy	RF. 31	Adicionar condición OrderBy
	RF. 32	Eliminar condición OrderBy
	RF. 33	Listar condición OrderBy

Requisitos no funcionales

Los requisitos no funcionales, como su nombre sugieren, no se refieren directamente a las funciones específicas que proporciona el sistema, sino a sus propiedades emergentes como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen restricciones del producto, como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (Sommerville, 2005).

- Requerimientos de apariencia o interfaz externa:
  - RNF1: la resolución mínima recomendada es de 1024x768 px.
  - RNF2: el sistema mostrará el nombre del producto.
  - RNF3: el sistema mostrará el logo del producto.
  - RNF4: el idioma que se utilizará será el español.
  - RNF5: el texto será de color negro.
- Requerimientos de usabilidad:
  - RNF6: los usuarios deberán poseer un conocimiento previo del manejo de una computadora personal.
  - RNF7: el usuario debe poseer conocimientos de diseño de base de datos y modelación de consultas.
- Requerimientos de portabilidad:
  - RNF8: la herramienta desarrollada deberá ser multiplataforma teniendo un correcto funcionamiento tanto en Linux como en Windows.
- Requerimientos de software:
  - RNF9: para que el sistema funcione deberá ser instalada la máquina virtual de java 7, para garantizar la gestión de los datos el SGBD PostgreSQL 9.2 y como IDE Netbeans 8.0.

- RNF10: sistema operativo Microsoft Windows 7 o superior o cualquier versión de sistemas Unix

- Requerimientos del diseño y de implementación:

  - RNF11: el sistema implementado será una aplicación de escritorio.
  - RNF12: el sistema será diseñado siguiendo los principios de programación orientada a objeto.
  - RNF13: el sistema se implementará usando NetBeans 8.0.
  - RNF14: el lenguaje de programación a utilizar será Java.

- Requerimientos de funcionalidad:

  - RNF15: el sistema mostrará los errores en forma de mensajes. Todos los mensajes de error del sistema deberán incluir una descripción textual del mismo.

2.3. Historias de usuarios

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas en las cuales el cliente describe brevemente las funcionalidades del sistema. 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 suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Canos, 2011).

Seguidamente se muestra la descripción de uno de los requisitos funcionales, mediante las historias de usuarios Guardar código DQL y SQL y Gestionar datos de prueba.

Tabla 2: Historia de usuario Guardar código DQL y SQL

Número: 6	**Nombre del requisito:** Guardar código DQL y SQL
Programador: Yoandrys León Batista	**Iteración Asignada:**3

Prioridad: baja	**Tiempo Estimado:**5 horas
Riesgo en Desarrollo:	**Tiempo Real:**3 horas
Descripción: *Permite guardar en un archivo (.txt) el código DQL y SQL de la consulta representada.*	
Observaciones: N/A	
Prototipo de interfaz:	

Tabla 3: Historia de usuario Gestionar datos de prueba

Número: 6	**Nombre del requisito:** Gestionar datos de prueba
Programador: Yoandrys León Batista	**Iteración Asignada:**3
Prioridad: Alta	**Tiempo Estimado:**21 horas
Riesgo en Desarrollo:	**Tiempo Real:**18 horas
Descripción: *Se encarga de realizar la gestión de los datos de prueba a las entidades previamente cargadas desde el modelo o desde una base de datos, teniendo en cuenta la relación existente entre estas, permitiendo la adición, modificación y eliminación de los mismos; así como exportar dichos datos a archivos (.php) para su uso posterior.*	
Observaciones: N/A	
Prototipo de interfaz:	

2.4. Planificación de iteraciones

Una vez identificados los requisitos y descritas las historias de usuario correspondientes, el equipo de desarrollo realiza una estimación del esfuerzo que supone requerirá la implementación de cada una, mientras el cliente define la prioridad en función del valor para el negocio. Finalmente, se llega a un acuerdo sobre el orden de implementación y el contenido

de las entregas, apostando por enfrentar las historias de más valor y riesgo lo antes posible (Canos, 2011).

La técnica de Planeación Poker permitió evaluar de forma rápida y eficaz la dificultad relativa de las historias de usuario especificadas. Para la aplicación de esta técnica cada miembro del equipo de desarrollo propone un valor de estimación para cada historia de usuario, el cual se interpreta como más complejo a mayor valor propuesto. La propuesta es un valor de la sucesión de Fibonacci, como propone la técnica, si estos no coinciden se exponen los argumentos de las estimaciones más alejadas y se repite el proceso hasta lograr un consenso (Cohn, 2006).

El esfuerzo estimado está dado en función de cuán difícil resultaría para el equipo cumplir con la tarea, no de la estimación del tiempo de implementación de la misma. La Tabla3 presenta el plan de iteraciones resultado de dicha planificación.

Tabla 4: Resultado de la Planificación

Número de iteración	**Historia de Usuario**	**Prioridad**	**Esfuerzo estimado**	
1	Gestionar entidades	Alta	13	42
	Gestionar condición Select	Media	5	
	Gestionar unión entre entidades	Media	3	
	Gestionar condición Where	Alta	21	
2	Gestionar condición GroupBy	Media	5	43
	Gestionar condición Having	Alta	8	
	Gestionar condición OrderBy	Baja	3	
	Construir consulta	Alta	21	
	Visualizar consulta DQL	Media	8	
	Visualizar consulta SQL	Media	8	
3	Gestionar datos de prueba	Alta	21	

	Visualizar resultado de la consulta	Baja	3	29
	Guardar código DQL	Baja	5	

2.5. Diseño

El diseño es la forma exacta en la que un requisito del cliente se puede convertir en un sistema o producto de software terminado. Crea una representación o modelo del software de forma detallada sobre la estructura de datos, la arquitectura, las interfaces y los componentes del software que son necesarios para implementar el sistema e influye directamente en la calidad del producto informático (Sanz Campos, 2012).

2.5.1. Diseño arquitectónico

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés) define al diseño arquitectónico de software, en el estándar STD 1471-2000 como: “la organización fundamental de un sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, así como los principios que orientan su diseño y evolución” (SA-1471-2000 2000).

Por otra parte la arquitectura es un nivel de diseño que se centra en la especificación de la estructura global del sistema (Cohn, 2006). En su forma más simple, la arquitectura es la organización de los componentes del programa (módulos), la manera en que estos componentes interactúan y la estructura de datos que utilizan (Kazman, y otros, 2003). Para el desarrollo de la solución se hace uso de la propia arquitectura (n-capas) definida en la edición estándar del lenguaje Java (Java SE).

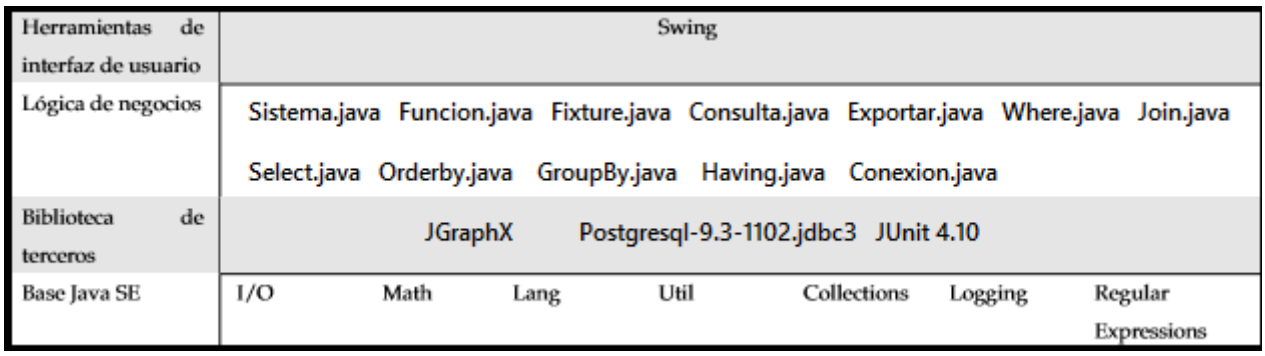

Figura 2: Vista de arquitectónica del sistema

La capa de más alto nivel corresponde a la interfaz de usuario, para la cual se utiliza swing, entorno que permite incorporar en las aplicaciones elementos gráficos de una forma versátil e intuitiva de conjunto con el uso de Netbeans IDE (García, y otros, 2000).

La siguiente capa corresponde a la lógica de negocio, abarca funcionalidades para la realización de consultas visuales y generación de archivos de datos. Esta capa da soporte a la interfaz de usuario y se sirve de servicios provistos por diferentes bibliotecas de código de terceros, así como bibliotecas base de la edición estándar de Java, las cuales corresponden a la tercera y cuarta capas respectivamente.

En correspondencia con lo anterior, la tercera capa incluye bibliotecas para la representación gráfica de las entidades y relaciones entre ellas y pruebas unitarias, entre otros.

La capa más baja abarca el conjunto de bibliotecas base de la versión estándar de Java, las cuales se utilizan en cada una de las capas antes mencionadas.

2.5.2. Modelado del diseño

Uno de los principios fundamentales de la metodología XP es la simplicidad, en la etapa de diseño se propone el empleo de tarjetas CRC (Clase Responsabilidad Colaboración), en lugar de diagramas de clases para la descripción del sistema en notación UML. La característica más sobresaliente de las tarjetas CRC es su sencillez y adaptabilidad (Casas, 2012). Dichas tarjetas representan cada una de las clases del sistema, en ellas se describen brevemente las responsabilidades de la clase y se listan las clases con las que colabora. Su utilización en el diseño potencia el uso de patrones de asignación de responsabilidades (Larman, 2004). A continuación, se muestran 2 de las tarjetas CRC generadas.

Tabla 5: Tarjeta CRC clase Consulta.java

Tarjeta CRC	
Clase	Consulta.java
Responsabilidades	**Colaboraciones**
1. Construir consulta y generar código en correspondiente en los lenguajes DQL y SQL.	Select.java Where.java

2. gestionar condición select 3. gestionar condición from 4. gestionar condición where 5. gestionar condición groupBy 6. gestionar condición having 7. gestionar condición orderBy	Having.java GroupBy.java OrderBy.java From.java

Tabla 6: Tarjeta CRC clase Fixture.java

Tarjeta CRC	
Clase	Fixture.java
Responsabilidades	**Colaboraciones**
Generar archivos de datos	Sistema.java

2.6. Patrones del diseño de software

Patrones GRASP

Una de las cosas más complicadas en la programación Orientación a Objeto consiste en elegir las clases adecuadas y decidir cómo estas clases deben interactuar. Incluso cuando utilizamos metodologías rápidas como XP y centramos el proceso en el desarrollo continuo, es complicado elegir cuidadosamente las responsabilidades de cada clase en la primera codificación y, fundamentalmente, en la refactorización de nuestro programa.

Por lo tanto, podemos decir que los patrones GRASP son una ayuda de aprendizaje que permiten al desarrollador a entender lo esencial del diseño de objetos y a aplicar el razonamiento del mismo de una forma metódica, racional y explicable. Este enfoque se entiende en el uso de los principios del diseño basado en patrones para la asignación de responsabilidades a las clases y objetos de una aplicación

Para el desarrollo de la solución no fue necesario definir la estructura sintáctica de los lenguajes DQL y SQL, para ello se concibió en la etapa de diseño el apoyo de los conceptos aportados en el patrón **intérprete**, no porque se deba analizar una cadena entrada por el

usuario, sino porque se debe validar la que se propone como salida, es decir, como código generado.

Dado el nivel de especificación de la distribución de funcionalidades en clases definidos por el patrón intérprete fu necesario aplicar otros patrones como:

- **Experto:** asigna la responsabilidad al experto en la información. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Durante todo el diagrama se manejó la información por cada entidad que le corresponde y se le asignó la responsabilidad de acuerdo a esa información por lo que se puede afirmar que el patrón está presente en todas las clases.
- **Controlador:** es un patrón que sirve como mediador entre una interfaz y el algoritmo que la implementa, recibiendo y procesando la información que recibe del usuario a través de dicha interfaz y haciéndola llegar al método que la emplea (método que fue llamado o instanciado) para dar la solución solicitada, que luego él se encargará de hacer llegar nuevamente al usuario. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control y se recomienda dividir los eventos del sistema en el mayor número de controladores posibles aumentando así la cohesión y disminuir el acoplamiento. Está presente en la clase **Sistema**:

```
29      private Tabla[] listTable;
30      private Consulta consulta;
31      private Relacion[] listRelacion;
32      private Entidad[] listEntidad;
33      private Fixture[] listFixtures;
```

Figura 3: Patrón Controlador

- **Creador:** el patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja

es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización, aportando bajo acoplamiento, mayor claridad, encapsulación y reutilización del diseño realizado. Este patrón está presente en la clase **Sistema**:

```
54          consulta = new Consulta();
55          con = new conexion();
```

Figura 4: Patrón Creador

Conclusiones parciales

Se determinaron de las prestaciones con las que contará la herramienta de forma general. Luego de aplicadas las técnicas de identificación de requisitos se obtuvieron un total de 33 requisitos funcionales, agrupados por la técnica de agrupación de requisitos teniendo en cuenta la relación entre las funciones que cumple en la solución propuesta, los cuales fueron especificados por el cliente en un total de 13 historias de usuarios en las que se realiza una descripción detallada de los mismos además se identificaron un total de 15 requisitos no funcionales. La especificación de las historias de usuario antes mencionadas permitió realizar el plan de iteraciones para el desarrollo de la solución quedando un total de 3 iteraciones en las que se especifican el número de historias de usuarios pertenecientes a cada iteración, así como el total de puntos estimados para cada una de estas. Luego de definida las iteraciones se procedió a realizar el diseño de la solución seleccionándose n-capas como arquitectura a emplear para el desarrollo de la misma, se realizaron las tarjetas CRC donde se especifican las responsabilidades y relaciones entre las clases definidas creándose un total de 12 tarjetas.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

3. Introducción

En el presente capítulo se realiza una descripción de las fases de codificación y prueba propuestas por la metodología XP. Se formaliza la estructura y precedencia de los operadores, en los lenguajes SQL y DQL, a utilizar en la solución. Además de describir el uso de patrones y estándares de codificación para garantizar la escalabilidad del código. También se presentan los resultados de la estrategia de pruebas propuesta por la metodología.

3.1. Implementación

La implementación del sistema consiste en la traducción del diseño en una plataforma tecnológica (lenguaje de programación, estándares de codificación y protocolos de comunicación). El resultado es un producto de software con las características descritas durante el diseño con las HU y los requisitos obtenidos, logrando un producto de software que puede ser utilizado por el cliente.

3.1.1. Estructura y precedencia de operadores

Desde el inicio de la implementación fue necesario concebir las reglas de los lenguajes con los que se trabaja para dar solución al problema que presenta en el presente trabajo. Como orden lógico operacional lo primero fue establecer las cláusulas contenedoras con las que se trabajaría, las cuales son:

- Select
- From
- Join
- innerJoin
- leftJoin
- where
- andWhere
- orWhere
- groupBy
- addGroupBy
- having
- andHaving
- orHaving
- OrderBya
- OrderBy.

Los operadores contenidos en estas son de igual importancia, acotando que no todas las cláusulas pueden contener a todos los operadores.

Operadores:

- EQ
- NEQ
- LT

- LTE
- GT
- GTE
- LIKE
- BETWEEN
- MAX
- MIN
- AVG
- COUNT.

Las cláusulas y operadores dispuestas en la presente solución fueron seleccionadas de entre las más empleadas del repositorio de consultas de uno de los proyectos de CEGEL. En la figura 5 se muestra la distribución de los operadores en los repositorios analizados.

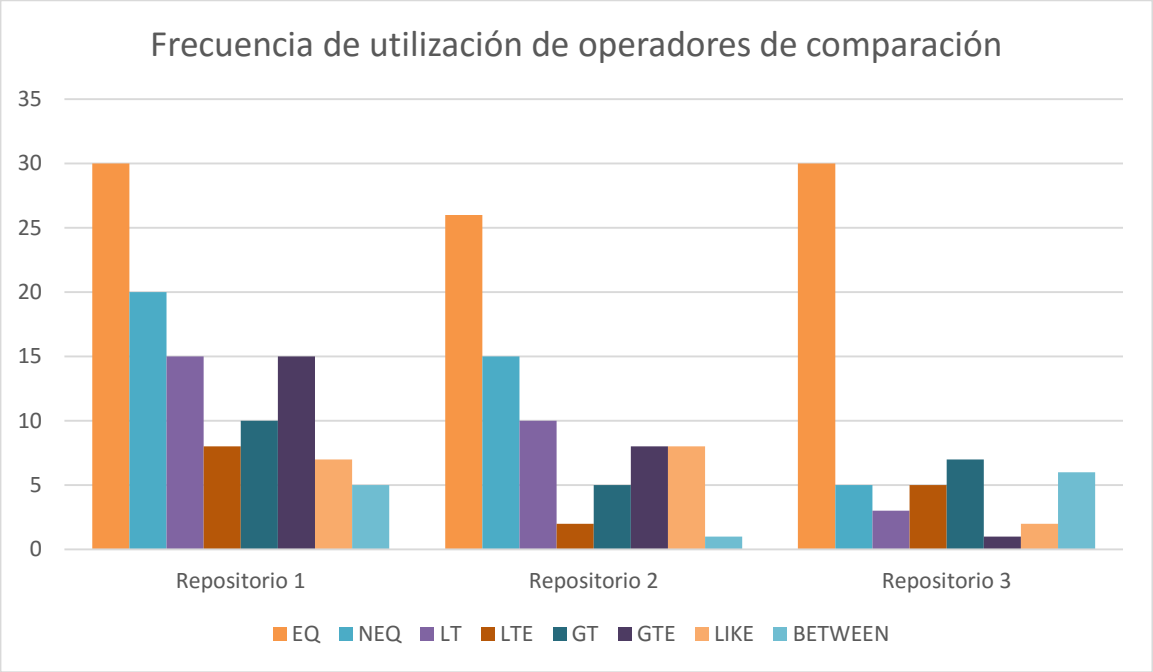

Figura 5: Distribución de los operadores

3.1.2. Orden y restricciones establecidas para la creación de consultas

En el diseño de clases realizado se tuvieron en cuenta algunas validaciones necesarias para garantizar que se cometieran la menor cantidad de errores posibles a la hora de modelar una consulta entre las que se encuentran las siguientes.

Tabla 7: Restricciones por cláusulas

Cláusulas	**Restricciones**
Select	Se debe de haber adicionado al menos una condición From.

From	Se debe de haber seleccionado alguna entidad
Where	Se debe de haber seleccionado alguna entidad y la misma debe aparecer en la cláusula From.
Groupby	Se debe de haber adicionado al menos una condición Select.
Having	Se debe de haber adicionado al menos una condición Groupby.
Orderby	En caso de haberse adicionado alguna condición Groupby solo se permitirán los atributos que aparezcan en la misma.

Para la construcción de las consultas se definió un orden lógico a seguir basado en las cláusulas utilizadas y los operadores permitidos en cada una de ellas el cual se presenta a continuación:

Tabla 8: Distribución de operadores

Cláusulas	**operadores**
where	1. EQ 2. NEQ 3. LT 4. LTE 5. GT

	6. TE 7. LIKE 8. BETWEEN,
having	1. MAX 2. MIN 3. AVG 4. SUM 5. COUNT 6. EQ 7. NEQ 8. LT 9. LTE 10. GT 11. GTE

3.1.3. Algoritmo de transformación de consultas SQL y DQL a partir del modelo

A partir de los análisis previos para identificar las cláusulas y operadores a emplear en las consultas a desarrollar y teniendo en cuenta las reglas establecidas en la documentación oficial de estos lenguajes se diseñó un algoritmo, el cual propone una serie de pasos a realizar para la obtención del código correspondiente al lenguaje especificado en función del modelo representado.

listCondiciones = lista de condiciones tipo Select, From, Where, GroupBy, Having, OrderBy.
consulta= código resultante
pos=posición inicial
lenguaje=lenguaje de consultas

Algoritmo 1. Generar código.

1: procedimiento generarCodigo(listCondiciones, consulta, pos, lenguaje)

2: arreglo de secuencia de cláusulas según orden [Select, From, Join, Where, GroupBy, Having, OrderBy]

3: mientras (pos<tamaño del arreglo de secuencia) hacer >

    5: si lenguaje=SQL hacer>

        6: consulta+=arreglo de secuencia [pos]

    7: fin si

    8: mientras (obtenerCondicion(arreglo de secuencia [pos], listCondiciones)) no esté vacía) hacer > se itera sobre cada una de las condiciones

        9: consulta+=condicion.obtenerCodigo(lenguaje)

        10: incrementar pos

    11: fin mientras

12: fin mientras

13: fin procedimiento

Reglas de transformación:

Cada cláusula define una forma de transformación en función del lenguaje y los operadores que contiene que, de forma general seria de la siguiente.

DQL

->Clausula (argumentos)

->addClausula (argumentos)

->orClausula (argumentos)

SQL

Clausula (argumentos)

El argumento de una cláusula es básicamente una expresión que puede estar compuesta por uno o varios operadores para los cuales se define la siguiente regla de transformación.

DQL

->expr()->Operador(parametros)

SQL

Valor1 Operador Valor2

Operador Valor1

Operador Valor1 and Valor2

3.1.4. Generación de archivos de datos

La generación de los archivos de datos se realiza a partir de una base de datos, teniendo en cuenta la relación existente entre las entidades, pertenecientes al modelo de datos sobre el que se está trabajando, con el objetivo de definir el orden en que deberían ser cargados dichos datos. Una vez establecido este orden se realiza la creación de los mismos a partir de sus correspondientes tablas en la base de datos.

Pasos para la generación de archivos de datos:

1. Selección de la entidad a la que se le desea generar el archivo de datos
2. Determinar relaciones y crear árbol de dependencias
3. Crear archivo de datos a partir de la información correspondiente en la base de datos según el orden definido en el árbol de dependencia

3.1.5. Estándares de codificación

Las buenas prácticas de programación indican el seguimiento de un conjunto de pautas desde la misma creación de los ficheros, lo cual facilita la trazabilidad y mantenibilidad del software.

Las clases serán colocadas en archivos independientes que solo contendrán el código de la misma. Se utilizará el estilo de codificación “UpperCamel Case”, el cual establece que los nombres iniciarán con letra mayúscula y de poseer más de una palabra, la primera letra de cada una deberá ser también mayúscula. No se permiten letras mayúsculas sucesivas a menos que se trate de siglas conocidas en el dominio del sistema (Binkley, 2009).En la Figura 3.0 se presenta un ejemplo en la declaración de la clase GroupBy.java.

```
 9   public class GroupBy {
10
11       String nombre;
12       String tabla;
13
14       public GroupBy(String nombre, String tabla) {
15           this.nombre = nombre;
16           this.tabla = tabla;
17       }
18
```

Figura 6: Utilización del estilo UpperCamel Case en declaración de clases

Los nombres de las variables deben ser descriptivos y concisos. No se usarán grandes frases ni abreviaturas. Se utilizará el estilo de codificación “LowerCamel Case” (Binkley, 2009), según

el cual los nombres iniciarán con letra minúscula y cada nueva palabra debe iniciar con mayúscula.

```
 8  import java.util.LinkedList;
 9
10  public class Consulta {
11
12      LinkedList<Select> listSelect;
13      LinkedList<From> listFrom;
14      LinkedList<Where> listWhere;
15      LinkedList<Join> ListJoin;
16      LinkedList<OrderBy> listOrdenBy;
17      LinkedList<GroupBy> listGroupBy;
18      LinkedList<Having> listHaving;
```

Figura 7: Utilización del estilo LowerCamel Case en los nombres de variables

Los nombres de las funciones deben dar una idea clara del objetivo con el que fueron concebidas. Al igual que para las variables, se utilizará el estilo “LowerCamel Case” (Binkley, 2009). La declaración del método getNombre() evidencia el empleo de este estilo.

```
20
21      public String getNombre() {
22          return nombre;
23      }
```

Figura 8: Utilización del estilo LowerCamel Case en la declaración de métodos

3.2. Pruebas

Los niveles de pruebas de software desarrolladas como parte del proceso de verificación y validación de la aplicación propuesta fueron: pruebas unitarias y pruebas de aceptación.

3.2.1. Pruebas unitarias

Se realizaron pruebas de unidad a las funcionalidades más significativas del sistema, con el propósito de verificar el adecuado desempeño de las mismas; para ello se utilizó el marco de trabajo jUnit en su versión 4.10, el cual viene integrado con el IDE Netbeans. Estas pruebas fueron realizadas para verificar el correcto funcionamiento del código antes de adicionarlo en la aplicación, y realizándolas nuevamente después de cada cambio realizado sobre estas y las que dependan de ellas. En las mismas se definen los casos críticos y el comportamiento

esperado de los métodos analizados (Ing. Oré , 2009), obteniendo resultados favorables en cada uno de ellas indicando la calidad de la solución desde el punto de vista funcional.

Para hacer uso de este marco de trabajo se definieron los siguientes pasos:

1. **Definición de los métodos a probar**

Las pruebas se realizaron solo a las funcionalidades críticas para el sistema, entre ellas están:

- + addWhere()
- + addFrom()
- + addSelect()
- + addHaving()
- + addGruopBy()
- + addOrderBy()
- + generarConsultaSQL()
- + generarConsultaDQL()
- + generarDatoDePrueba()

2. **Realización de las pruebas**

Los resultados de las pruebas son almacenados dentro de una lista. Luego, el marco de trabajo JUnit comprueba que cada uno de los resultados obtenidos coincide con los resultados esperados y muestra en una ventana los resultados obtenidos, cuando estos son satisfactorios para todas las pruebas realizadas, se observa una línea verde en la ventana, en caso contrario aparece una línea roja. Se desarrollaron un total de 12 pruebas con el JUnit, distribuidas en dos iteraciones. Se obtuvieron en la primera un total de 3 errores que fueron subsanados. En la última se obtuvieron resultados satisfactorios para cada prueba realizada. En la siguiente figura aparecen los resultados obtenidos.

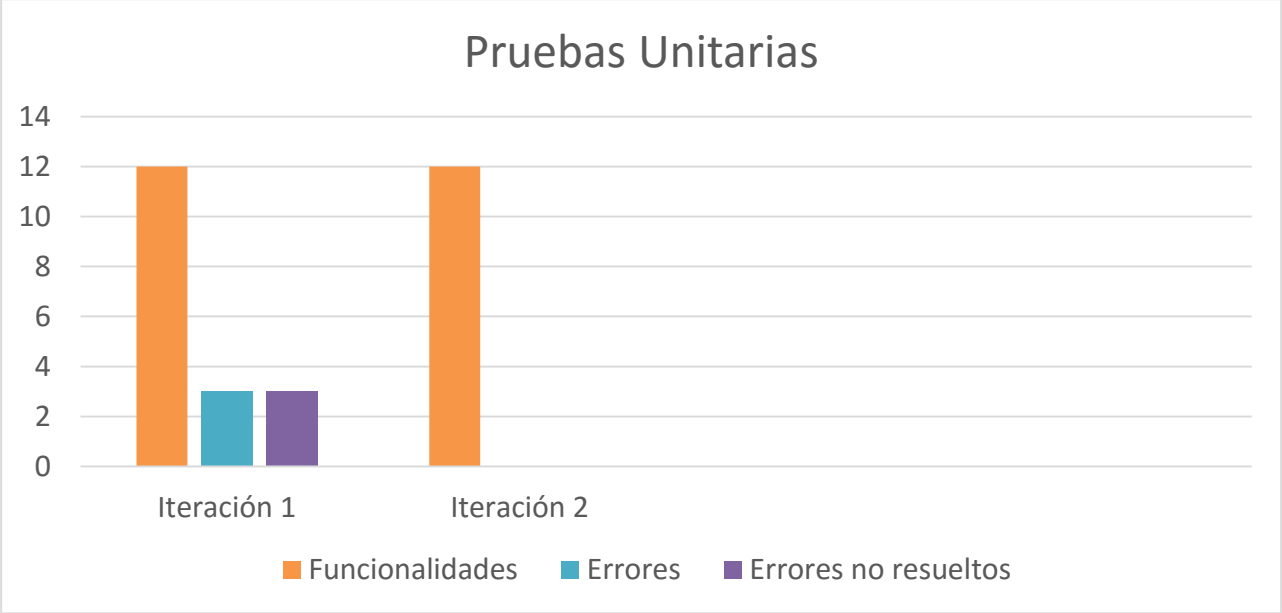

Figura 9: Resultados de las pruebas unitarias

3.2.2. Pruebas de aceptación

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido, estás se dividen según la metodología empleada en pruebas alfa y pruebas beta. A continuación, se muestra uno de los casos de pruebas llevados a cabo durante las pruebas alfa:

Tabla 9: Caso de prueba de aceptación de la HU Gestionar condición where

CASO DE PRUEBA DE ACEPTACIÒN		
Historia de Usuario: HU_11	Gestionar condición Where	
Descripción:	Permite adicionar, eliminar y listar condiciones where a la consulta que se desee desarrollar.	
Condiciones de ejecución:	El usuario debe entrar al menos un punto.	
Escenarios de prueba:	**Flujo del escenario:**	**Resultados esperados:**

EP:1	Adicionar where con datos válidos.	Se introducen datos válidos y se acciona el botón “+”.	Se muestra la nueva condición en la lista de condiciones
EP:2	Adicionar where con datos incorrectos.	Se introducen datos incorrectos y se acciona el botón “+”.	Se muestra un mensaje de error en una ventana emergente indicando el error: “Datos incorrectos”.
EP:3	Adicionar where con datos incompletos.	Se acciona el botón “+” sin haber completado los campos necesarios para la ejecución de la operación.	Se muestra un mensaje de error en una ventana emergente indicando el error: “Campos vacíos”.

En la Figura 10 se muestra la relación entre las pruebas satisfactorias y no satisfactorias, evidenciando una evolución positiva en cuanto a la corrección de deficiencias y eliminación de errores en la aplicación. En cada iteración no solo se comprobaron los requisitos planificados..

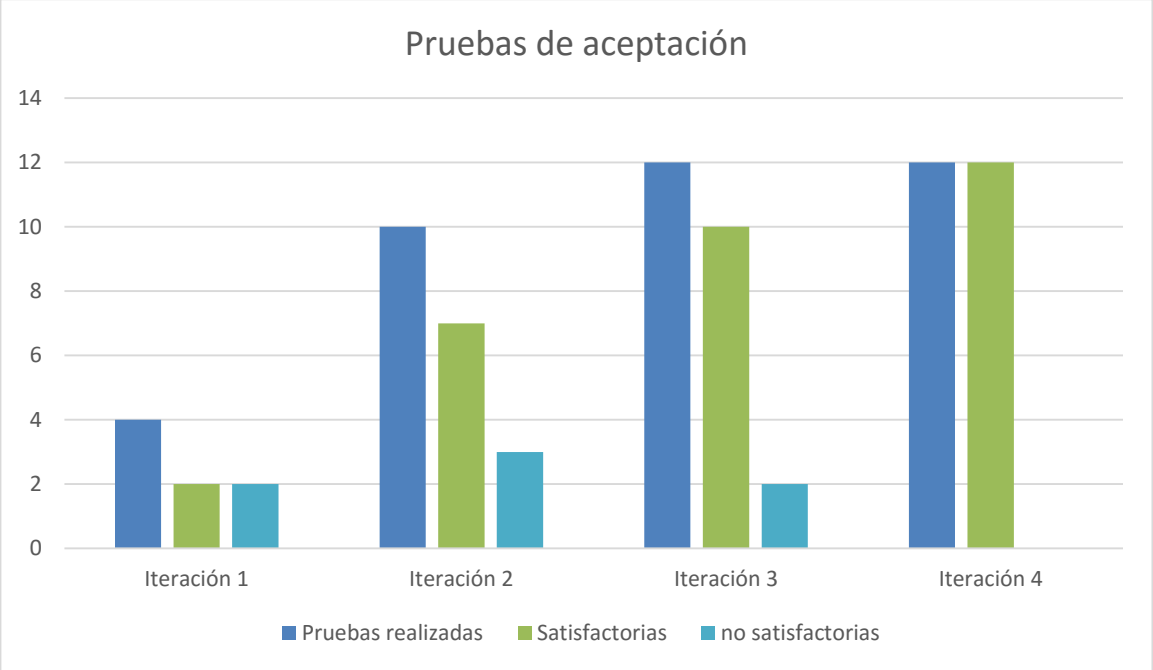

Figura 10: Resultados de las pruebas de aceptación

3.2.3. Validación del objetivo de la investigación

Para la validación de la variable de la investigación se definieron dos etapas, en la primera se comprobó el grado de satisfacción y completitud de las consultas generadas con la solución que se propone en el presente trabajo, según juicio de expertos, en función de esto se seleccionaron 5 especialistas de CEGEL, los cuales evaluaron las consultas generadas por la herramienta de forma satisfactoria, además con el fin de comprobar la reducción del tiempo empleado en la elaboración de dichas consultas, utilizando la herramienta y sin el uso de esta, se sometieron a evaluación consultas de baja y media complejidad incluyendo a personas con diferentes niveles de conocimiento en el área, arrojando que se logra una disminución en el tiempo empleado para la realización de estas de 3 a 4 minutos en caso de las consultas de baja complejidad y de 5 a 7 minutos en caso de las consultas de media complejidad. En la figura 11 se muestran los resultados obtenidos.

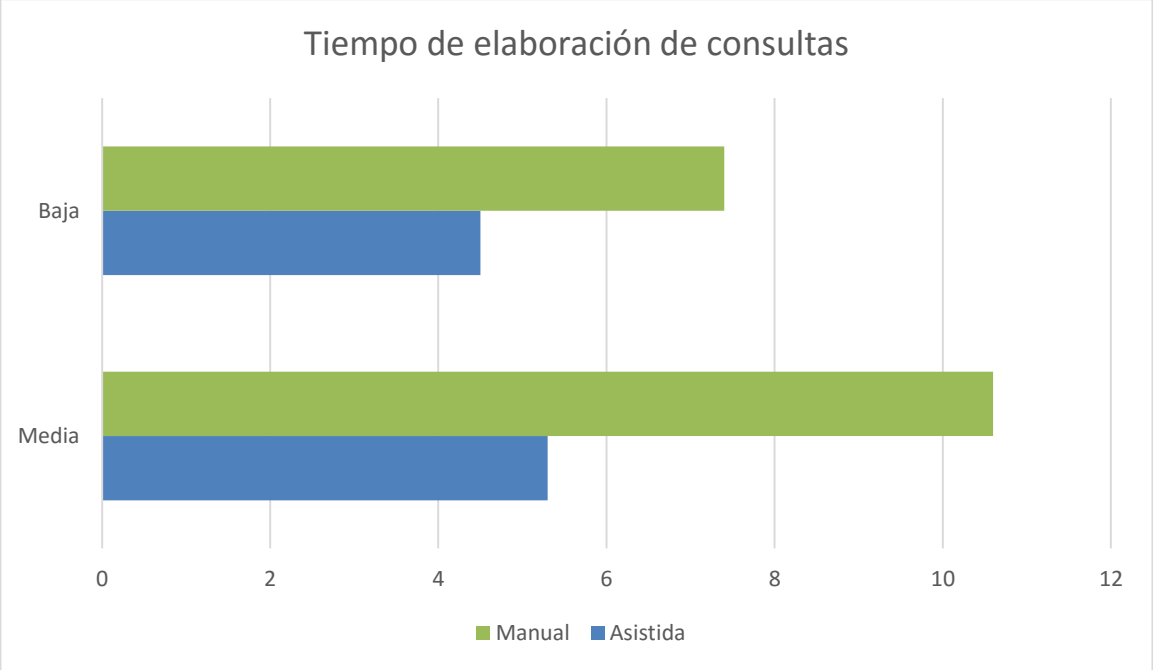

Figura 11: Resultados de la comparación de los tiempos empleado en la realización de las consultas

Durante la segunda etapa se validó la reducción del tiempo empleado por los especialistas en la generación de archivos de datos, para ello se evaluaron un total de 5 entidades con un numero de atributos relativamente pequeño, obteniéndose que el tiempo de desarrollo de forma manual es relativamente elevado en comparación con el tiempo de elaboración asistido por la herramienta, con un diferencia de 17 minutos para los casos que no son de complejidad elevada y sin exigir la incorporación de mucha información referente a los mismos, lo que evidencia que el tiempo de realización manual, tienda a incrementarse cuando el conjunto de datos aumenta su complejidad, no siendo así con la herramienta, por lo que se logra una disminución en el tiempo de realización de los mismos. En la figura 12 se muestran dichos resultados.

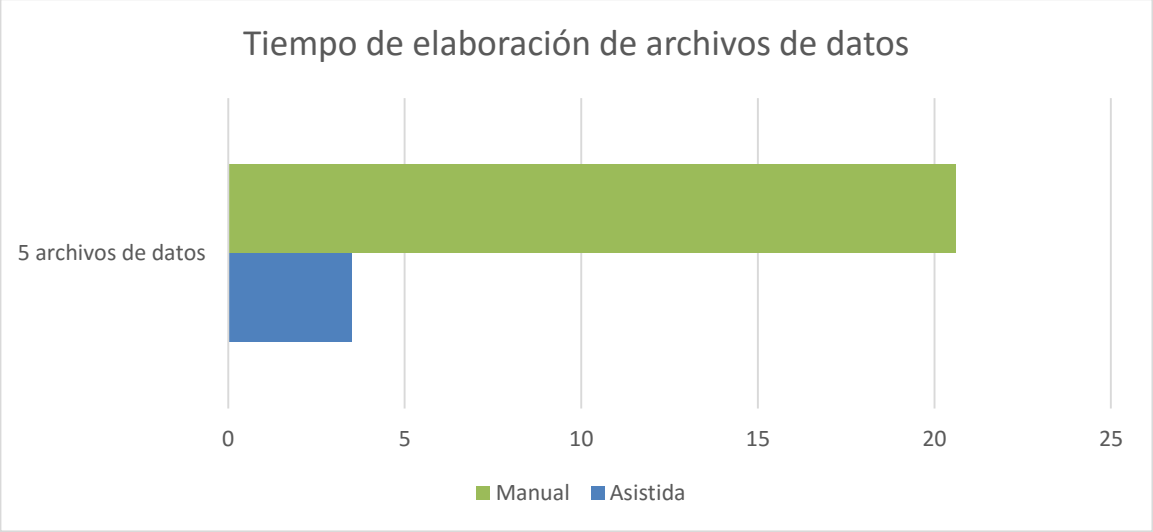

Figura 12: Resultados de la comparación de los tiempos empleados para la elaboración de archivos de datos

.

Conclusiones parciales

Se realizó un correcto análisis de los operadores y cláusulas a emplear para la realización de las consultas DQl y SQL, así como los algoritmos de transformación para la generación del código correspondiente a cada uno de estos lenguajes y el conjunto de pasos a seguir para la adecuada generación de archivos de datos. Fueron seleccionados los estándares de codificación a emplear en la solución, determinándose UpperCamel Case para la declaración de las clases, así como LowerCamel Case para la declaración de variables y métodos. Además, se definieron los operadores y cláusulas a emplear en la solución según el análisis realizado en los repositorios provistos por el cliente. Se estableció la precedencia y jerarquía entre las cláusulas, así como qué operadores soporta cada una, también un conjunto de restricciones y validaciones con vistas a disminuir los errores en la modelación de las consultas. Como resultado de todo esto se obtuvo un sistema informático que permite la modelación de consultas visuales, así como la creación de archivos de datos. Finalmente se desarrollaron las pruebas automatizadas empleando el marco de trabajo Junit, luego se probó el nivel de aceptación con las pruebas alfa y beta, definida por la metodología que guía el presente trabajo de investigación, eliminando todas las no conformidades detectadas. El conjunto de pruebas de aceptación realizadas con el cliente, así como las de rendimiento (tiempo de elaboración de consulta y tiempo de elaboración de archivos de datos) demostraron que se logra una disminución en el tiempo empleado para la realización de dichos procesos.

CONCLUSIONES GENERALES

La investigación realizada cumple con los objetivos planteados mediante el desarrollo de ORMaping y se arriba a las siguientes conclusiones:

- El análisis de los referentes teóricos y de los sistemas informáticos estudiados evidenció la necesidad de desarrollar un sistema que minimice el tiempo en la generación de consultas y archivos de datos.
- Se logró actualizar el estado del arte en función de los principales conceptos, términos y herramientas empleadas durante la investigación y el desarrollo de la propuesta de solución.
- La metodología de desarrollo elegida (XP) posibilitó la obtención del modelo de diseño de la solución propuesta.
- Se logró desarrollar un sistema informático que permite la modelación de consultas visuales, así como la creación de archivos de datos.
- La validación de la investigación se realizó a partir de la aplicación de técnicas, métricas y pruebas que garantizaron el correcto funcionamiento del sistema y demostraron la satisfacción del cliente hacia el sistema desarrollado.
- Finalmente se logró desarrollar una solución que permite la realización visual de consultas y archivos de datos, disminuyendo el tiempo requerido para las tareas de los arquitectos de datos de CEGEL.

RECOMENDACIONES

Se recomienda para futuras iteraciones y versiones del sistema propuesto:

- Ampliar el soporte para las conexiones a otros SGBD y otras herramientas ORM.
- Permitir la creación de consultas de mayor complejidad que permitan la utilización de las funciones no contempladas en la presente propuesta.

REFERENCIAS BIBLIOGRÁFICAS

Bauer, Christian and King, Gavin. 2004. *Hibernate in Action. Practical Object/Relational Mapping.* s.l. : Manning Publications, 2004.

Beck, Kent. 2000. *Extreme Programming Explained: Embrace Change.* 2000.

Berriero, Pablo Sánchez. 2011. *Mantenimiento de software.* 2011.

Binkley, D. 2009. IEEE. [Online] 2009. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5090039..

Canos, J & P. Letelier. 2011. *Metodologías Ágiles en el Desarrollo de Software. Universidad Politécnica de Valencia.* Valencia : s.n., 2011.

Carrero , Angel. 2013. Conceptos básicos de ORM (Object Relational Mapping). *Programación en castellano.* [Online] 2013. http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349.

Casas, S. & H. Reinaga. 2012. *Aspectos tempranos: un enfoque basado en tarjetas crc. Sociedad Colombiana de Computación.* Colombia : s.n., 2012.

CEGEL. Centro de Gobierno Electrónico. Suite de Gestión de Proyecto. [Online] [Cited: Febrero 10, 2015.] http://gespro.cegel.prod.uci.cu.

Cohn, M. 2006. *Agile Estimating and Planning.* 2006.

Editorbfb. 2011. Qué es un entorno de desarrollo integrado, IDE. *Programación Desarrollo.* [Online] 2011. http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/.

Feriñas, Hernández. 2013. *Versión Miranda R2 del módulo de adquisición del SCADA Guardian del Alba.* s.l. : Universidad de las Ciencias Informáticas, 2013.

Fernandez, Oscar Belmonte. 2004. *Introducción al lenguaje de programación Java. Una guía básica.* España : s.n., 2004.

García, Javier, Ignacio Rodríguez and Íñigo., José. 2000. *Aprenda java como si estuviera en primero. Escuela Superior de Ingenieros Industriales.* 2000.

Guerrero, Carlos A. and Suárez, Johanna M. y Gutiérrez, Luz E. 2013. *Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web.* [Online] Revista Información Tecnológica Vol. 24 no.3, 2013. [Cited: 03 25, 2015.] La Serena, Colombia.. http://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci_arttext. ISSN 0718-0764 .

Guibert Estrada, Lisandra y Altuna Castillo, Enrique José. 2011. *Ingeniería de requisitos del software educativo "Mis Mejores Cuentos". Centro de Tecnologías para la Formación. Universidad de las Ciencias Informáticas.* [Online] Centro de Tecnologías para la Formación, Universidad de las Ciencias Informáticas., 2011. [Cited: 03 14, 2015.] La Habana, Cuba. http://gte2.uib.es/edutec/sites/default/files/congresos/edutec11/Ponencias/Mesa%205/Guibert%20-%20Altuna-%20Edutec%202011.F.pdf.

Ing. Oré , Alexander. 2009 . UNIT TESTING - PRUEBAS UNITARIAS. *CalidadySoftware.com .* [Online] 2009 . http://www.calidadysoftware.com/testing/pruebas_unitarias2.php .

jGraphx. 2006. jGraphx User Manual. *jGraphx User Manual.* [Online] 2006. [Cited: Febrero 10, 2016.] https://jgraph.github.io.

Kazman, R. and Bass., L. 2003. *Software Architecture in Practice.* 2003.

Khan, Iqbal M. 2005. Develop high speed .NET and Java applications. *Alachisoft.* [Online] noviembre 25, 2005. [Cited: noviembre 25, 2014.] http://www.alachisoft.com/resources/articles/orm.html.

Larman, Craig. 2004. *UML Y PATRONES.Introducción al análisis y diseño orientado a objetos y al proceso unificado.* España : s.n., 2004.

Letelier, P. & Penade M. C. 2003. *Metodologías Ágiles en el Desarrollo de Software: eXtreme Programming(XP).* Valencia : Universidad Politécnica de Valencia , 2003.

Malfará, Dayvis et al. 2006. *Testing en extreme programming. Gestión de Software.* 2006.

Montenegro Jiménez, Isaac and Rodríguez Rodríguez, Luis y Salazar Bermúdez, Gabriela. Agosto/Diciembre 2012. *Uso de patrones de diseño de software: un enfoque práctico.* [Online] Revista Semestral de la Universidad de Costa Rica. Volumen 22 Número 2., Agosto/Diciembre 2012. [Cited: 03 23, 2015.] San José, Costa Rica. ISSN: 1409-2441 .

NetBeans. 2014. *NetBeans.* [Online] 2014. [Cited: 12 07, 2015.] http://netbeans.org/index_es.html.

Pacheco, Nacho. 2011. *Doctrine 2 ORM Documentation.* 2011. sf2-es.net16.net/_downloads/Doctrine2ORM.pdf.

PostgreSQL. 2003. PostgreSQL. *PostgreSQL.* [Online] 2003. [Cited: febrero 10, 2016.] https://www.postgresql.org/about/.

Pressman, Roger S. 2003. *Ingeniería del Software. Un enfoque práctico. Sexta Edición.* s.l. : Mc Graw Hill, 2003. 970-10-5473-3.

Ricart, Oridalmis and Sarmiento, Carlos Andréz. 2015. *Aplicación informática para realizar el mapeo relacional de clases del marco de trabajo Doctrine.* Habana : s.n., 2015.

Rodríguez Tello, Eduardo A. 2012. *Estrategias y técnicas de prueba del software.* [Online] CINVESTAV, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional., 2012. [Cited: 04 15, 2015.] Tamaulipas, México. www.tamps.cinvestav.mx/~ertello/swe/sesion15.pdf.

Sánchez, Jorge. 2004. *Diseño Conceptual de Base de Datos.Guía de aprendizaje.* California : s.n., 2004.

Sanz Campos, Dayana y Rueda Flores, Jorge Alberto. 2012. *Diseño e implementación del módulo "Inscripción" del Sistema para la Gestión de Antecedentes Penales (SIGESAP).* Universidad de las Ciencias Informáticas. La Habana, Cuba. : s.n., 2012. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. http://bibliodoc.uci.cu/RDigitales/2012/ diciembre/3/TD_05408_12.pdf.

Sommerville, Ian. 2005. *Ingeniería del software. Séptima Edición.* Madrid. España : Pearson Educación. S. A., 2005. 84-7829-074-5.

Team, Doctrine. 2016. http://www.doctrine-project.org. *Doctrine-Project.* [Online] 2016. [Cited: Febrero 16, 2016.] http://www.doctrine-project.org.

Tecnología. 2014. *Lenguajes de Programación.* [Online] 2014. [Cited: 12 02, 2014.] http://www.areatecnologia.com/informatica/ lenguajes-de-programacion.html.