

**Universidad de las Ciencias Informáticas**

**Facultad 3**



*Aplicación informática para realizar el mapeo relacional de clases del marco de trabajo Doctrine*

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

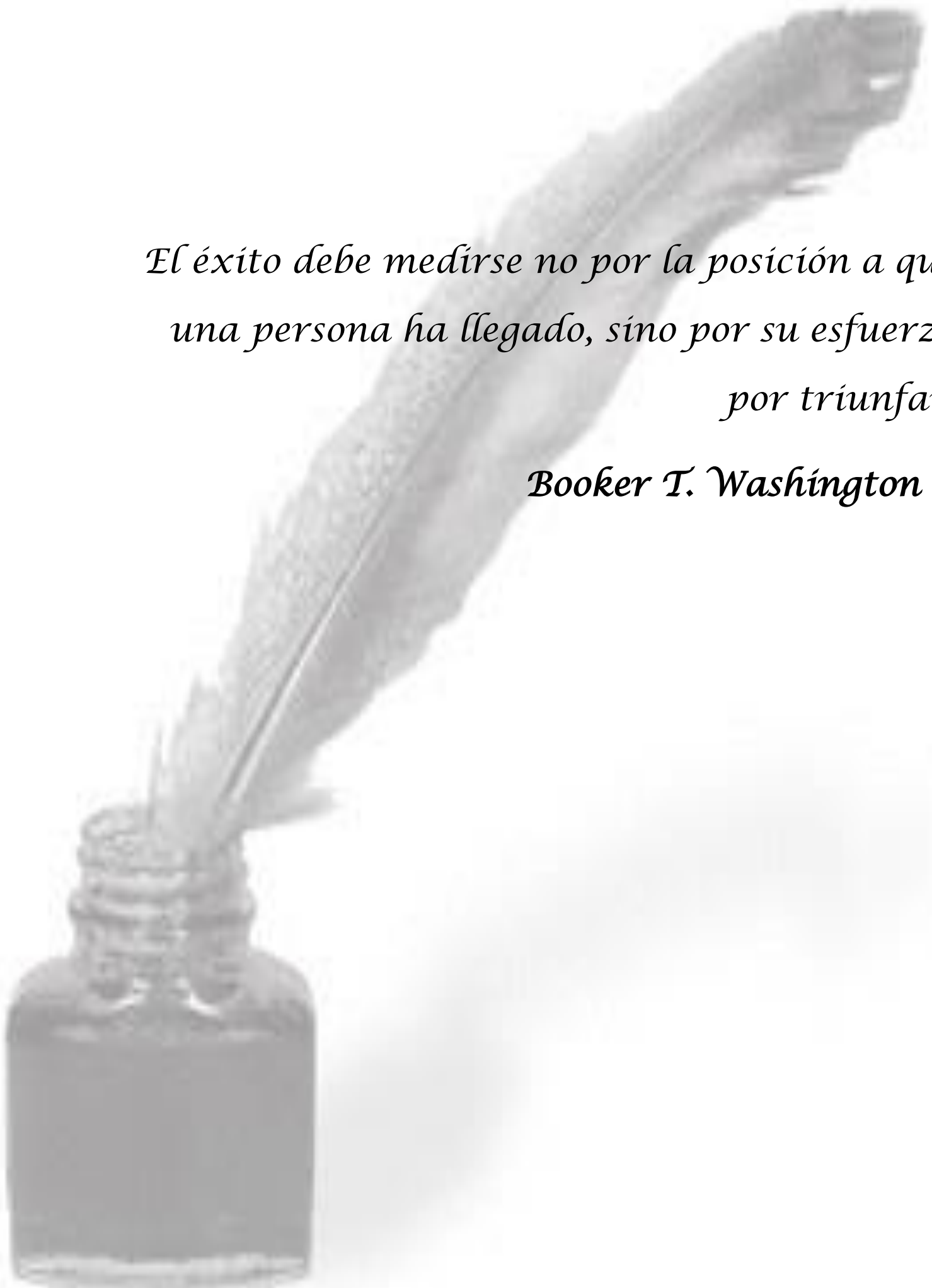
**Autores:**

Oridalmis Ricart Rodríguez  
Carlos Andrés Sarmientos Claro

**Tutores:**

Ing. José Carlos Pupo Acosta  
Ing. Manuel Álvarez Alonso  
Ing. Abel Andres Irsula Tumbarell

La Habana, Junio de 2015

A black and white photograph of a quill pen resting in a glass inkwell. The quill is positioned diagonally, pointing towards the top right corner of the frame. The inkwell is a simple, rounded glass bottle with a narrow neck. The background is a plain, light color, possibly white or light gray, which makes the dark tones of the quill and inkwell stand out.

*El éxito debe medirse no por la posición a que  
una persona ha llegado, sino por su esfuerzo  
por triunfar.*

*Booker T. Washington*

# Declaración de autoría

---

---

## Declaración de autoría

Se declara que Oridalmis Ricart Rodríguez y Carlos Andrés Sarmiento Claro son los únicos autores del trabajo de diploma *Aplicación informática para realizar el mapeo relacional de clases del marco de trabajo Doctrine* y se le reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

---

Firma de la autora

Oridalmis Ricart Rodríguez

---

Firma del autor

Carlos Andrés Sarmientos Claro

---

Firma del Tutor

Ing. José Carlos Pupo Acosta

---

Firma del tutor

Ing. Abel Andres Irsula Tumbarell

---

Firma del tutor

Ing. Manuel Álvarez Alonso

## *Datos de contacto*

---

### **Datos de tutor:**

**Nombre:** Ing. Abel Andres Irsula Tumbarell

Ingeniero en Ciencias Informáticas.

**Correo electrónico:** aairsula@uci.cu

### **Datos de Tutor:**

**Nombre:** Ing. Manuel Álvarez Alonso

Ingeniero en Ciencias Informáticas.

**Correo electrónico:** malvareza@uci.cu

### **Datos de Tutor:**

**Nombre:** Ing. José Carlos Pupo Acosta

Ingeniero en Ciencias Informáticas.

**Correo electrónico:** jcpupo@uci.cu

### **Datos del Autor:**

**Nombre:** Oridalmis Ricart Rodríguez

**Correo electrónico:** oricart@estudiantes.uci.cu

### **Datos del Autor:**

**Nombre:** Carlos Andrés Sarmiento Claro

**Correo electrónico:** casarmiento@estudiantes.uci.cu

# *Dedicatoria*

---

**Oridalmis**

**De corazón a:**

***A quienes siempre me enseñaron a luchar por alcanzar mis sueños y confiaron en mi: Mis padres Orialy y Emilio.***

***A mi hermana Oriality por apoyarme en todo momento y darme su amor.***

***A mis abuelos y mi familia por entregarme su amor y su apoyo para terminar la carrera con resultados satisfactorios. ¡Los quiero mucho y gracias a todos !***

**Carlos**

***Este trabajo va dedicado a las personas más importantes de mi vida***

***A mis padres, hermanos y familiares por siempre estar presentes en mi vida.***

# *Agradecimientos*

---

*Oridalmis*

*A la Revolución y en especial a nuestro Fidel por existir y hacer realidad este sueño.*

*A mis padres por haberme brindado todo su apoyo, su amor y la mejor educación.*

*A mi hermana que la quiero con la vida y siempre estará en mi corazón.*

*A toda mi familia que siempre me ayudó en todo lo que necesité durante los 5 años de estudio.*

*A mi guapísimo novio Leo por su apoyo incondicional , su paciencia, su amor por casi dos años y sobre todo por que lo amo.*

*A Mercy y Pacho por acogerme en su casa durante toda la carrera como una hija más.*

*Al piquete del grupo 3504 porque construimos una familia en todos estos años.*

*A los tutores por tanto apoyo que me brindaron y confianza que depositaron en mí para culminar esta gran tarea.*

*A mi gran amiga Yeni y su mamá por darme todo su amor y cariño cuando lo necesité.*

*A mis amigas Eli y Dayi por apoyarme en todo momento.*

*A todos los que de una u otra forma ayudaron a la realización de este trabajo.*

*A mi compañero de tesis por aguantarme todo este tiempo.*

*Finalmente, a todas las personas dentro y fuera de la uci que durante mi carrera aportaron un granito de arena para culminar mis estudios.*

*Carlos*

*Agradecer profundamente a mis padres, familiares y hermanos por apoyarme incondicionalmente en todo momento.*

*Agradecer a todos mis compañeros y amigos por compartir los buenos y malos momentos.*

*Agradecer a mis tutores por apoyarme cuando más lo necesite.*

*A todas las personas que formaron parte de mi vida durante estos cinco años.*

*A mi compañera de tesis por apoyarme todo este tiempo.*

En el proceso de desarrollo de software generalmente se utiliza la técnica de programación orientada a objeto y el sistema de base de datos de tipo relacional. La diferencia entre los dos paradigmas induce a los programadores a convertir los objetos del lenguaje de programación a registros de la base de datos y de igual forma la operación inversa. Precisamente para resolver este problema los desarrolladores se apoyan en las herramientas de mapeo relacional de objetos. La inexistencia de una herramienta que facilite el trabajo de los arquitectos de datos del Centro de Gobierno Electrónico en sus tareas en términos de tiempo de realización de las mismas, promovió la construcción de la aplicación ORM MAPPING. 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 mapeo de grandes volúmenes de clases del marco de trabajo Doctrine, comprobado mediante la validación del sistema a través de la prueba piloto aplicada 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 ,el lenguaje de programación Java y la librería AWT con el componente Swing, además de la guía ofrecida por la metodología de desarrollo ágil Extreme Programming.

**Palabras claves:** base de datos relacionales, herramientas de mapeo, mapeo relacional de objetos, metodología.

---

---

## Índice

Introducción.....	1
Capítulo 1: Fundamentación teórica.....	5
1.1    Introducción .....	5
1.2    Mapeo relacional de objetos. Fundamentos de sus componentes.....	5
1.2.1    Descripción del proceso ORM.....	5
1.3    Doctrine.....	8
1.4    Análisis crítico de las herramientas de apoyo al proceso ORM .....	9
1.4.1    Herramienta de modelado Visual Paradigm for UML 8.0.....	9
1.4.2    ORM Designer .....	10
1.5    Metodologías de desarrollo .....	11
1.6    Naturaleza del software.....	15
1.7    Patrones de diseño .....	16
1.8    Herramientas y lenguajes de programación .....	16
1.8.1    Entorno de desarrollo integrado (IDE) .....	16
1.8.2    Sistemas gestores de base de datos (SGBD) .....	17
1.8.3    Lenguajes de programación.....	18
1.8.4    Tecnologías empleadas .....	19
1.9    Pruebas de Caja Blanca.....	19
1.10    Pruebas de Caja Negra.....	20
1.11    Conclusiones del capítulo .....	20
Capítulo 2. Diseño e implementación .....	21
2.1    Introducción .....	21
2.2    Descripción de la solución.....	21
2.3    Especificación de los requisitos del software.....	22
2.3.1    Requisitos funcionales .....	22
2.3.2    Requisitos no funcionales .....	24
2.3.3    Validación de requisitos .....	25
2.4    Descripción de la arquitectura.....	26



---

---

2.5	Fases del proceso de desarrollo .....	28
2.5.1	Fase de exploración .....	28
2.5.2	Fase de planificación.....	32
2.6	Fase de diseño.....	35
2.6.1	Patrones de diseño .....	35
2.6.2	Tarjetas de Clase, Responsabilidad y Colaboración (CRC).....	38
2.6.3	Diagrama de clases del diseño .....	38
2.6.4	Validación del diseño .....	41
2.7	Fase de implementación .....	46
2.7.1	Tareas de Ingeniería .....	46
2.7.2	Estándares de codificación .....	49
2.8	Prototipo de interfaz de la aplicación.....	51
2.9	Conclusiones del capítulo .....	52
Capítulo 3.	Pruebas y validación de las variables de la investigación.....	53
3.1	Introducción .....	53
3.2	Fase de pruebas .....	53
3.2.1	Prueba de caja blanca.....	53
3.2.2	Prueba de caja negra.....	56
3.2.3	Prueba para requerimientos de eficiencia .....	59
3.3	Validación de la investigación .....	60
3.4	Conclusiones del capítulo .....	60
Conclusiones generales	.....	61
Recomendaciones.....	.....	62
Referencias bibliográficas .....	.....	63

## Índice de figuras

Figura 1. Proceso de Mapeo Relacional de Objetos (7).....	6
Figura 2. Arquitectura por capas. (Elaboración propia).....	27
Figura 3. Ejemplo de uso del patrón Creador. (Elaboración propia).....	36
Figura 4. Ejemplo de uso del patrón Alta Cohesión. (Elaboración propia) .....	37
Figura 5. Ejemplo de uso del patrón Fábrica Abstracta. (Elaboración propia).....	38
Figura 6. Diagrama de clases del diseño. (Elaboración propia) .....	40
Figura 7. Representación de la evaluación de la métrica TOC para el atributo responsabilidad. (Elaboración propia) .....	42
Figura 8. Representación de la evaluación de la métrica TOC para el atributo complejidad. (Elaboración propia) .....	43
Figura 9. Representación de la evaluación de la métrica TOC para el atributo reutilización. (Elaboración propia) .....	43
Figura 10. Representación de la evaluación de la métrica RC en el atributo Acoplamiento. (Elaboración propia) .....	44
Figura 11. Representación de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. (Elaboración propia) .....	45
Figura 12. Representación de la evaluación de la métrica RC en el atributo cantidad de pruebas. (Elaboración propia) .....	45
Figura 13. Ejemplo de código aplicando el comentario. (Elaboración propia) .....	50
Figura 14. Ejemplo de código aplicando estándares de codificación. (Elaboración propia) .....	50
Figura 15. Ejemplo de código aplicando estándares de codificación. (Elaboración propia) .....	51
Figura 16. Interfaz principal de la aplicación ORM MAPPING. (Elaboración propia).....	52
Figura 17. Código generado con JUnit en Netbeans. (Elaboración propia).....	55
Figura 18. Resultados de las pruebas con Junit. (Elaboración propia) .....	56
Figura 19. Caso de prueba Adicionar Entidad. (Elaboración propia).....	57
Figura 20. Resultado de las pruebas de Caja Negra. (Elaboración propia).....	58

## Índice de tablas

Tabla 1. Características convergentes entre las herramientas .....	11
Tabla 2. Comparación entre metodologías ágiles y tradicionales (21) .....	12
Tabla 3. Comparación entre las principales metodologías ágiles (22) .....	13
Tabla 4. Comparación entre software web vs software de escritorio .....	15
Tabla 5. Personas relacionadas con el sistema.....	22
Tabla 6. HU Exportar a ORM.....	28
Tabla 7. HU Importar ORM.....	29
Tabla 8. HU Sincronizar BD .....	30
Tabla 9. Estimación del esfuerzo por HU.....	32
Tabla 10. Plan de duración de iteraciones.....	33
Tabla 11. Plan de entregas .....	34
Tabla 12. Descripción Tarjeta CRC Sistema .....	38
Tabla 13. Descripción de las clases del diagrama de diseño.....	41
Tabla 14. Responsabilidad.....	42
Tabla 15. Complejidad.....	42
Tabla 16. Reutilización .....	43
Tabla 17. Acoplamiento.....	44
Tabla 18. Complejidad de Mantenimiento .....	44
Tabla 19. Cantidad de pruebas .....	45
Tabla 20. Tarea de Ingeniería #1 .....	46
Tabla 21. Tarea de Ingeniería #2 .....	46
Tabla 22. Tarea de Ingeniería #3 .....	47
Tabla 23. Tarea de Ingeniería #4 .....	47
Tabla 24. Tarea de Ingeniería #5 .....	48
Tabla 25. Tarea de Ingeniería #6 .....	48
Tabla 26. Tarea de Ingeniería #7 .....	48
Tabla 27. Métodos para las pruebas unitarias .....	54
Tabla 28. Descripción de las variables del requisito Adicionar entidad .....	57
Tabla 29. Resultados del tiempo de respuesta.....	59

## Introducción

El uso de tecnologías de la información y las comunicaciones se ha convertido en un componente central de toda empresa o negocio que busque desarrollar software con calidad en el menor tiempo posible. Este objetivo puede ser cumplido con la utilización de sistemas de base de datos encargados de persistirlos para un posterior uso, permitiendo el procesamiento de información para realizar una efectiva toma de decisiones. Se define base de datos 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 (1).

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 están 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 (2). Entre los diferentes modelos de base de datos que existen se encuentran los modelos conceptuales, los lógicos y los físicos (1). Siendo las bases de datos relacionales un ejemplo representativo del modelo lógico, estas son las más comunes y extendidas, ya que permiten modelar problemas reales mediante las relaciones entre tablas o entidades, lo cual es su función fundamental.

Herramientas como los frameworks o marcos de trabajo, facilitan el trabajo con las bases de datos en el desarrollo de aplicaciones informáticas. Precisamente la utilización de los mismos ha reportado beneficios considerables en cuanto a reducción de tiempos y errores durante la fase de implementación en el proceso de desarrollo (3). Estos, emplean la técnica de programación denominada Mapeo-Relacional-Objeto (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<sup>1</sup> que definen el mapeo entre los objetos y la Base de Datos”* (4).

En la actualidad, es común la utilización de marcos de trabajo que realicen ORM para facilitar el acceso a los datos. En el ámbito de las aplicaciones web uno de los ORM que más gana adeptos para su uso 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.

La Universidad de las Ciencias Informáticas (UCI), es uno de los motores principales en el desarrollo de productos informáticos en Cuba. Está concebida en facultades, las cuales cuentan con centros de desarrollo donde se llevan a cabo procesos complejos con avanzadas herramientas y técnicas.

---

<sup>1</sup> Metadatos: datos que describen otros datos.

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 (5). En aras de alcanzar tales fines, se emplean los marcos de trabajo para realizar la persistencia de la información en bases de datos relacionales.

La realización de esta tarea de forma ágil y eficiente requiere, necesariamente, dominio en el manejo del marco de trabajo. El poco acceso de los arquitectos de datos de CEGEL a licencias válidas para el software privado de mapeo existente, dificulta su uso y soporte. Asimismo, el proceso de mapeo de grandes volúmenes de clases en negocios con alto nivel de complejidad, conlleva a que las tareas de los arquitectos de datos se vean afectadas respecto al tiempo de realización de estas, al tener que ejecutar este proceso de forma manual, una clase a la vez. Igualmente el proceso de cambio y actualización de las entidades se divide en dos momentos distintos, en las clases y la base de datos, por lo que provoca un consumo mayor de tiempo para culminar la tarea.

En correspondencia con el contexto descrito, se identificó el siguiente **problema a resolver**: las insuficiencias presentes en el proceso de mapeo de grandes volúmenes de clases del marco de trabajo Doctrine, atentan contra el tiempo de realización de las tareas de los arquitectos de datos de CEGEL.

Este problema se enmarca en el **objeto de estudio**: proceso de mapeo relacional de objetos del marco de trabajo Doctrine, identificándose como **campo de acción**: aplicaciones informáticas para el proceso de mapeo relacional de objetos del marco de trabajo Doctrine.

Para resolver el problema identificado se definió el siguiente **objetivo general**: desarrollar una aplicación informática que favorezca el mapeo de grandes volúmenes de clases del marco de trabajo Doctrine, 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 al objetivo 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. Definición de la arquitectura a utilizar en el desarrollo de la solución.
5. Implementación de la solución a partir de las herramientas seleccionadas y los requisitos identificados.
6. Aplicación de pruebas al software que permitan validar la solución desarrollada.
7. 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 uso de una aplicación informática para realizar el mapeo de grandes volúmenes de clases del marco de trabajo Doctrine, 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 mapeo relacional de objetos del marco de trabajo Doctrine y de las herramientas informáticas que existen en la actualidad para ello.
- **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 mapeo relacional de objetos del marco de trabajo Doctrine en CEGEL, que posteriormente son sometidos a verificaciones empíricas.
- **Modelación:** permitió 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 mapeo relacional de objetos del marco de trabajo Doctrine en CEGEL y los problemas asociados al mismo.

- **Medición:** con el objetivo de obtener información numérica acerca del proceso de mapeo relacional de objetos del marco de trabajo Doctrine en CEGEL. Además, se utiliza para evaluar el comportamiento del software diseñado y su pertinencia con los requisitos establecidos.

## **Estructuración por capítulos**

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 relacional de objetos, 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. Diseño e implementación** se hace una descripción general de la solución propuesta de desarrollo del sistema, así como la realización del levantamiento de los requisitos funcionales y no funcionales, entrada al diseño del software. También se presenta la arquitectura con la que se desarrolló el mismo y se hace referencia a los patrones de diseño empleados. Además, se muestran los artefactos definidos por la metodología de desarrollo de software seleccionada y se describe la construcción de la aplicación, explicando los aspectos principales de la implementación.

En el **Capítulo 3. Pruebas y validación de las variables de la investigación** se hace el diseño y ejecución de pruebas sobre la aplicación en busca de errores relacionados con su funcionalidad. Por último se muestran los resultados obtenidos del empleo de la prueba piloto realizada por los especialistas del centro, para la validación de la investigación.

## Capítulo 1: Fundamentación teórica

### 1.1 Introducción

En el presente capítulo se estudian los conceptos relacionados con el proceso de mapeo relacional de objetos del marco de trabajo Doctrine. Se realiza un análisis de las herramientas de apoyo a este proceso para conformar un sólido basamento teórico. Además, se explica la selección de las herramientas y tecnologías empleadas para el desarrollo de la solución propuesta.

### 1.2 Mapeo relacional de objetos. Fundamentos de sus componentes

Actualmente, la industria del software se enfrenta con los problemas inherentes causados por la integración de las siguientes tecnologías: los modelos orientados a objetos y las bases de datos relacionales, distribuyendo así las responsabilidades de modelar la lógica y persistir los objetos. Es entonces cuando cobra importancia el empleo de herramientas ORM y con ello la realización del proceso de mapeo relacional de objetos, ya que se necesita de una interfaz que traduzca la lógica de los objetos a la lógica relacional, la cual está formada por objetos que permiten acceder a los datos y que contienen en sí mismos el código necesario para hacerlo (6).

#### 1.2.1 Descripción del proceso ORM

El proceso ORM consiste en crear una capa de abstracción intermedia entre el SGBD y la aplicación. El mismo permite crear un acceso a datos robusto e independiente de un SGBD específico. En CEGEL, una tarea común de cualquier arquitecto de datos es desarrollar el acceso a datos de los sistemas, para ello necesita utilizar algunas herramientas que en su conjunto le permiten realizar dicha actividad. Por tanto, para poder unificar las bases de datos y los conceptos de orientación a objetos, la aplicación se apoya en el uso de marcos de trabajo de mapeo, en este caso Doctrine para las aplicaciones web, como se muestra en la figura 1.



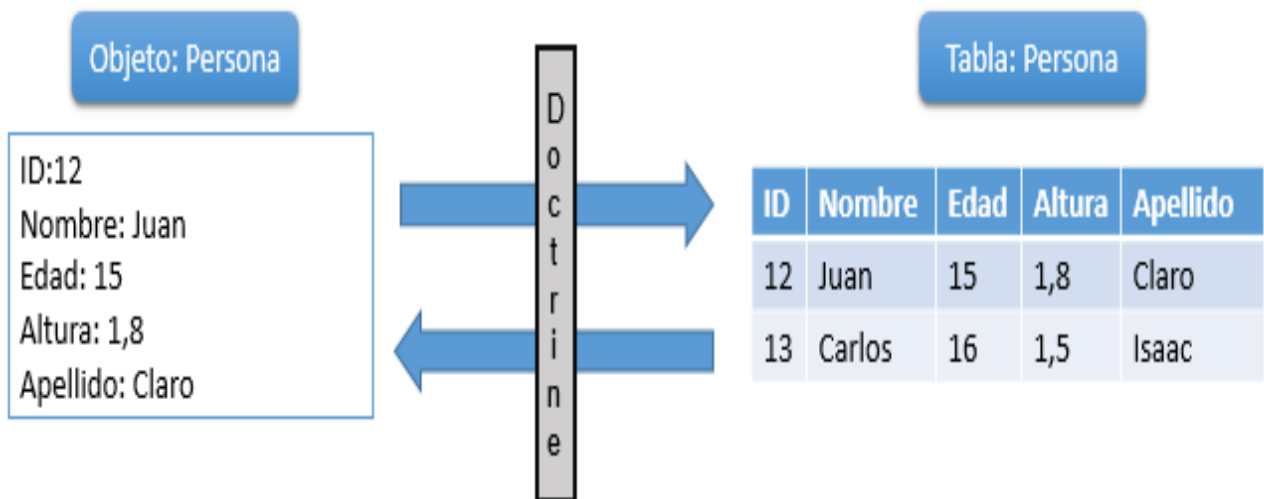


Figura 1. Proceso de Mapeo Relacional de Objetos (7)

El flujo normal del proceso de mapeo para la creación de un nuevo módulo en el centro, se realiza de la forma siguiente:

- Los diseñadores del sistema en conjunto con los arquitectos de datos elaboran el diagrama de clases persistentes haciendo uso de Visual Paradigm.
- Se genera la base de datos, ya sea por conexión al SGBD o exportándola a un fichero con extensión **.sql**.
- Se generan las clases php con sus respectivos atributos y se cargan en el IDE de desarrollo para su edición.
- Se editan cada una de las clases obtenidas y se les incorporan las anotaciones php de Doctrine 2.

A la hora de realizar las modificaciones al modelo ocurre un proceso similar al descrito anteriormente:

- El Comité de Control de Cambios del proyecto aprueba la modificación al modelo.
- Entre los diseñadores del sistema y los arquitectos de datos se elabora la modificación tratando de optimizar y minimizar el impacto del cambio.
- Se actualiza el diagrama de clases persistentes.
- Se editan las entidades a modificar.
- Se editan las tablas en base de datos que representan las entidades modificadas.

Para darle solución a los problemas existentes en el centro a la hora de realizar el mapeo relacional de objetos, se tienen en cuenta las siguientes características que brindan los ORM.

- Rapidez en el desarrollo: la mayoría de las herramientas permiten la creación del modelo por medio de la lectura del esquema de tabla y relación de la base de datos de forma automática. Esto evita el tiempo de codificación repetitiva por parte del programador y los errores que puede traer en la implementación (7).
- Abstracción del motor de base de datos: permite separar totalmente del sistema de base de datos que se utilice y si en el futuro se debe cambiar de motor de bases de datos, no afectará al sistema (7).
- Interfaz simple: proporciona una interfaz simple para el manejo de objetos a través de su propio lenguaje de consulta (8).

Los mismos ofrecen varias ventajas que facilitan su uso en la práctica, las cuales se mencionan a continuación:

- Reutilización: permite llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- Encapsulamiento: encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
- Mantenimiento del código: la correcta ordenación de la capa de datos permite modificar y mantener el código (9).

Existen distintos tipos de herramientas ORM en correspondencia con el lenguaje de programación que se desee utilizar para el desarrollo de determinado software, como es en el caso de Java con Hibernate, en .NET se usa ADO.NET Entity Framework y en PHP es muy utilizado Doctrine. Este último, es una herramienta que apoya el desarrollo de diversas aplicaciones y es notable la cantidad de desarrolladores que lo están adoptando, ya que ha reportado beneficios considerables y por estar en un constante crecimiento en cuanto a documentación y miembros de su comunidad (10). Para entender cómo esta herramienta apoya todo el proceso ORM, es necesario definir este concepto.

## Herramienta ORM

Es una herramienta de modelado y generación de código que se conecta a la base de datos y lee su esquema, luego le permite asignar objetos a las tablas de bases de datos y vistas, operaciones de inserción, actualización, selección y eliminación, consultas y llamadas a procedimientos almacenados. También permite definir relaciones de uno-a-uno, uno-a-muchos, muchos-a-uno y muchos-a-muchos entre objetos en función de las relaciones entre las tablas de la base de datos (11).

Actualmente se tiene un conocimiento previo de las características del proceso y se conoce que existen herramientas capaces de realizarlo. A continuación, se explica detalladamente todo lo relacionado con el tema que se estudia en cuestión.

## Marco de trabajo

Marco de Trabajo del inglés Framework, se define como "un conjunto de componentes físicos y lógicos estructurados de tal forma que permiten ser reutilizados en el diseño y desarrollo de nuevos sistemas de información" (12). Los propios autores definen que los marcos de trabajo contienen patrones y buenas prácticas que apoyan el desarrollo de un producto y un proceso con calidad. Lo anterior permite vislumbrar la importancia de adoptar un marco de trabajo y cómo su selección debe ser una de las actividades relevantes al inicio de todo proceso de desarrollo de software.

### 1.3 Doctrine

Doctrine es un marco de trabajo de ORM para *PHP* 5.3.0 o versiones superiores, el cual proporciona persistencia transparente de objetos PHP, situándose en la parte superior de una poderosa capa de abstracción de base de datos. La principal tarea de los ORM para PHP es la traducción transparente entre objetos y las filas relacionales de la base de datos (13).

Una de las principales características de Doctrine es su lenguaje de consulta estructurado (SQL, por sus siglas en inglés) llamado Lenguaje de Consulta de Doctrine (DQL, por sus siglas en inglés), inspirado en el Lenguaje de Consulta de Hibernate (HQL, por sus siglas en inglés) y el bajo nivel de configuración que se necesita para comenzar un proyecto. Además DQL difiere ligeramente 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, Obtener, Actualizar y Borrar (CRUD - Create, Retrieve, 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, PostgreSQL) (13).

Características principales de Doctrine (14):

- 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 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 más 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 (15).

- 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.

Actualmente, existen algunas herramientas que realizan ORM, las cuales serán analizadas seguidamente para obtener un sólido basamento teórico del tema que se estudia.

## 1.4 Análisis crítico de las herramientas de apoyo al proceso ORM

### 1.4.1 Herramienta de modelado Visual Paradigm for UML 8.0

Visual Paradigm for UML (VP) es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar documentación (16).

Entre sus características fundamentales se tienen (16):

- Multiplataforma: soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
- Interoperabilidad: intercambia diagramas y modelos con otras herramientas. Soporta la importación y exportación a formatos XMI y XML y archivos Excel. Permite importar proyectos de Rational Rose y la integración con Microsoft Office Visio.
- Ingeniería de código: permite la generación de código e ingeniería inversa para los lenguajes: Java, C, C++, PHP, XML, Python, C#, VB .Net, Flash, ActionScript, Delphi y Perl.
- Integración con entornos de desarrollo: apoyo al ciclo de vida completo de desarrollo de software en IDEs como: Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, Jbuilder y otros.

A pesar de que la herramienta brinda disímiles características que pueden ser aprovechadas por diferentes usuarios, la misma presenta algunas desventajas que impiden su utilización:

- Tiempo utilizado en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto.
- Generación de clases ORM en lenguaje PHP para Doctrine en un formato complejo de mantener por los desarrolladores.
- No permite sincronizar los cambios en el modelo de clases con el código y la base de datos generados con anterioridad.

## 1.4.2 ORM Designer

ORM Designer es una herramienta de diseño de base de datos visual con el apoyo total de los marcos ORM. Se ha creado para reemplazar kilobytes de definiciones de texto con interfaz gráfica fácil de usar. Gracias a un diseño muy flexible, casi cualquier marco ORM con definiciones almacenadas en un lenguaje de marcas puede ser totalmente compatible (17).

Características de la herramienta (17):

- Interfaz de usuario: está construido en una interfaz de usuario diseñada para la creación de modelos de datos.
- Manipulación del modelo: herramientas avanzadas de búsqueda, que son capaces de buscar a través de las relaciones, las columnas, las regiones y las tablas.
- Los archivos y serialización: se basa en un documento XML muy flexible que se utiliza para almacenar definiciones del modelo de datos y los datos de visualización.

La herramienta brinda múltiples características que favorecen su utilización, pero presenta algunos inconvenientes para el desarrollo de aplicaciones en la universidad por ser un software privado, los cuales se mencionan a continuación:

- Restricciones en el uso (marcadas por licencias).
- Imposibilidad de redistribución.
- El costo de la aplicación es elevado.
- Escasa documentación técnica para su uso.
- El soporte de la aplicación es exclusivo del propietario (18).

A continuación se realiza un análisis de las herramientas antes mencionadas en función de algunas de las características que se prevén en la solución propuesta de la investigación.

La solución propuesta considera las características particulares siguientes:

- **C1:** multiplataforma.
- **C2:** no posee licencia de usuario final.

- **C3:** satisfacer las características distintivas del proceso ORM para los arquitectos de datos de CEGEL.
- **C4:** sincronización desde el modelo en la aplicación hacia la base de datos.

Tabla 1. Características convergentes entre las herramientas

Características /Herramientas	C1	C2	C3	C4
Visual Paradigm	X			
ORM Designer	X		X	X

Las herramientas presentadas constituyen una base para el diseño e implementación de la propuesta, no para su utilización, debido a que no cumplen con algunas de las características principales requeridas por los arquitectos de datos, así como su dificultad de acceso. Esto queda reflejado en las limitaciones que presentan, al no cumplir en su totalidad con C2, C3, C4. Por lo tanto, se decidió desarrollar una aplicación bajo software libre que permitirá disminuir el tiempo invertido por los arquitectos de datos, en la creación y mantenimiento del mapeo de las clases de acceso a datos de Doctrine.

El estudio de las herramientas de apoyo al proceso ORM, permitió identificar un conjunto de funcionalidades básicas, así como componentes esenciales que debe poseer la herramienta a desarrollar. Estas aplicaciones presentan características particulares que aunque resuelven problemas para diferentes usuarios que las utilizan, no lo realizan directamente para los arquitectos de datos.

Ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software, surgen las metodologías de desarrollo.

## 1.5 Metodologías de desarrollo

Se define metodología de desarrollo como un marco de trabajo que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. En la presente investigación se decide adoptar esta definición considerando la filosofía de desarrollo de Pressman. Existen dos grupos en los cuales se dividen estas metodologías: los métodos tradicionales y los procesos ágiles, los cuales tienen marcadas diferencias (19).

Para la elección de alguno de los perfiles anteriores es necesario identificar elementos que representen las condiciones, necesidades e intereses del equipo de desarrollo, dentro de los cuales se identifican (ver Tabla 2):

- Tiempo de desarrollo
- Tamaño del equipo de desarrollo
- Comunicación entre los miembros del equipo de desarrollo
- Capacidades y habilidades de desarrollo
- Intensidad en el trabajo en equipo

Tabla 2. Comparación entre metodologías ágiles y tradicionales (21)

Indicadores	Metodologías de desarrollo de software	
	Ágiles	Tradicionales
Comunicación entre los miembros del equipo	Alta	Media
Tamaño del equipo de desarrollo	Pequeño	Grande
Tiempo de desarrollo	Limitado	Amplio
Capacidades y habilidades	Alta	Media
Intensidad en el trabajo en equipo	Alta	Media

El perfil de una metodología ágil se identifica más con las necesidades y condiciones del proyecto a desarrollar que una tradicional. La elección queda enmarcada fundamentalmente por las características del equipo de desarrollo concordando con el criterio de Pressman de *que “los defensores del desarrollo ágil del software resaltan la importancia de los factores de las personas en un desarrollo ágil exitoso”* (20).

A partir de la elección de una metodología ágil para guiar el proceso de desarrollo de software, se analizan varios exponentes de la misma, Extreme Programming (XP), Scrum y Crystal Methodologies, por ser de las más conocidas que han sido citadas y explicadas en libros de Ingeniería de Software (21). Para ello se toman en cuenta un conjunto de elementos que se muestran en la siguiente tabla:

- Participación del cliente
- Realización de pruebas

- Nivel de documentación
- Nivel de flexibilidad para el equipo de trabajo

Tabla 3. Comparación entre las principales metodologías ágiles (22)

Indicadores	XP	Scrum	Crystal
<b>Participación del cliente</b>	Activa, retroalimentación continua entre el cliente y el equipo de desarrollo.	En cada iteración	Activa
<b>Pruebas</b>	Ejecutadas constantemente ante cada modificación del sistema.	En cada iteración	Diariamente
<b>Documentación</b>	Se sustituye la documentación por la comunicación.	Poca	Poca
<b>Centra la atención</b>	Programadores	Gestión del proyecto	Programadores
<b>Flexible para el equipo</b>	Poco	Poco	Poco

El equipo de desarrollo seleccionó la metodología XP por las siguientes ventajas que ofrece con respecto a las demás, la misma le da solución a los problemas derivados del cambio en los requerimientos del software, lo cual trae consigo el aumento en la productividad. XP da la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Además, sigue la idea de la programación en pares, dada las ventajas que ofrece esta respecto a la creación del código, pues se pueden evitar errores y malos diseños al controlar cada línea de código y decisión de diseño instantáneamente. La interacción entre ambos desarrolladores puede generar discusiones que lleven a mejores estructuras y algoritmos, aumentando la calidad del software. También permite realizar pruebas unitarias ejecutadas constantemente ante cada modificación de la aplicación, lo cual garantiza la elaboración de una solución estable y disponible durante su desarrollo. Por otra parte, esta metodología fomenta la retroalimentación entre el equipo de desarrollo y el cliente, permitiendo que este último pueda ser consultado en cualquier momento del proceso de desarrollo (22).

## XP

De las metodologías ágiles, XP es la más popular en la actualidad. XP es una metodología 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. Esta metodología se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, en la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios.

## **Las Historias de usuario (HU)**

Las HU son el artefacto utilizado en XP para especificar los requisitos del software. Su tratamiento es muy dinámico y flexible, en cualquier momento estas pueden reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada una es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (22).

Respecto a la información contenida en las mismas, existen varias plantillas sugeridas pero no existe un consenso al respecto, sin embargo una ya reconocida es en la que pueden identificarse los siguientes contenidos: número de historia, prioridad técnica y del cliente, riesgo, estimación técnica, descripción (23). Además, al comienzo de cada iteración estarán registrados los cambios y según esto se planificará la siguiente iteración. Las HU son descompuestas en tareas de ingeniería y asignadas a los programadores para ser implementadas durante una iteración.

## **Fases**

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de vida ideal de XP se divide en fases: Exploración, Planificación, Diseño, Implementación y Pruebas (22).

### **Fase I: Exploración**

Los clientes plantean a grandes rasgos las HU 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 (22).

### **Fase II: Planificación**

El cliente establece la prioridad de cada HU y recíprocamente, 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 (22).

### **Fase III: Diseño**

Es el proceso para definir la arquitectura, los componentes, las interfaces y otras características de un sistema o un componente. Es muy importante, ya que se establecen los mecanismos, para que este sea revisado y mejorado de manera continua a lo largo del proyecto, según se van añadiendo funcionalidades al mismo. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto (24).

## Fase IV: Implementación

Se genera todo el código fuente de las especificaciones recogidas en las HU definidas para la solución, de manera que se cumplan los requisitos de todo el sistema. XP plantea que el mejor artefacto que se puede generar es el código, su buena calidad influirá en gran medida en el producto final. La misma es influenciada por los estándares de codificación los cuales serán tenidos en cuenta en la fase (22).

## Fase V: Pruebas

Uno de los pilares de la metodología XP es el uso de pruebas para comprobar el funcionamiento del código que se vaya implementando. Al no ser demasiado extensas las distintas funcionalidades de la aplicación, no se harán pruebas que analicen partes de las mismas, sino que las pruebas se realizarán para las funcionalidades generales que debe cumplir el programa especificado en la descripción de requisitos (22).

## 1.6 Naturaleza del software

El equipo de desarrollo decidió implementar una aplicación de escritorio en lugar de una aplicación web teniendo en cuenta los valores de los indicadores: rendimiento, seguridad y portabilidad (25), comparándolos en función del nivel de procesamiento que debe tener la misma para importar y exportar los ficheros de mapeo. Los resultados arrojados en la siguiente tabla comparativa, favorecen el uso de una aplicación de escritorio. Todas las características se adaptan satisfactoriamente a los requerimientos de los desarrolladores del centro.

Tabla 4. Comparación entre software web vs software de escritorio

Características	Aplicación Web(Navegador)	Escritorio
Rendimiento	Su tiempo de respuesta es más lento.	El tiempo de respuesta es muy rápido.

Seguridad	Es responsabilidad del proveedor de servicio.	Es responsabilidad del administrador de la empresa y de cada usuario que usa el sistema de forma local.
Portabilidad	Total dependencia del navegador web.	Siempre funcionará, sin dependencia del navegador web.

## 1.7 Patrones de diseño

A la hora de desarrollar software es importante el uso de patrones de diseño, brindando una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. *“Los patrones de diseño son descripciones estructurales y funcionales de cómo resolver, de forma concreta un determinado problema mediante orientación a objetos”* (26). Permiten que el diseño sea más flexible, elegante y reutilizable.

- **Patrones** General Responsibility Assignment Software Patterns (GRASP, 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 (26).
- **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 (26).

## 1.8 Herramientas y lenguajes de programación

### 1.8.1 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 (27).

## NetBeans 8.0

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 (28).

Características principales que permitieron emplearlo en la solución (28):

- Le da 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 frameworks.
- Posee un editor de código robusto, multilenguaje, con el habitual coloreado y sugerencias de código, acceso a clases pinchando 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.
- Optimización de código: por su parte el Profiler ayuda a optimizar las aplicaciones e intentar hacer que se ejecuten más rápido y con el mínimo uso de memoria.

### 1.8.2 Sistemas gestores de base 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 (29) (1).

#### PostgreSQL 9.1

**PostgreSQL:** es un SGBD relacional orientado a objeto y libre, publicado bajo la licencia BSD. 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 del uso de bloqueos explícitos (30) (31).

## **PgAdmin 1.14.1**

Es una aplicación gráfica para administrar PostgreSQL, siendo la más completa y popular con licencia de código abierto. Es capaz de gestionar versiones a partir de la PostgreSQL7.3 y superiores ejecutándose en cualquier plataforma. El PgAdmin3 está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y facilita enormemente la administración. Una característica interesante de pgAdmin3 es que, cada vez que se realiza alguna modificación en un objeto, escribe la(s) sentencia(s) SQL correspondiente(s), lo que hace que, además de una herramienta muy útil, sea a la vez didáctica. También incorpora funcionalidades para diseñar consultas (32).

## **1.8.3 Lenguajes de programación**

### **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 (33).

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) y la máquina virtual de Java en su versión 7 (33) (34).

### **PHP 5.3**

PHP es un acrónimo recursivo que significa PHP Hypertext Pre-processor es un lenguaje interpretado de propósito general ampliamente usado, diseñado especialmente para desarrollo web y que puede ser introducido dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser

desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno (35) (36).

## SQL

Lenguaje de consulta estructurado (SQL, por sus siglas en inglés), es un lenguaje surgido de un proyecto de investigación de IBM para el acceso a bases de datos relacionales. Actualmente se ha convertido en un estándar de lenguaje de bases de datos y la mayoría de los SGBD lo soportan, desde sistemas para ordenadores personales, hasta grandes ordenadores. Este es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de alto nivel o de no procedimiento, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros y no a registros individuales, permite una alta productividad en codificación (37) (38).

### 1.8.4 Tecnologías empleadas

#### Librería AWT

*AWT* es el acrónimo del X Window Toolkit para Java, donde X se definió por Sun como *Abstracto*. Se trata de una biblioteca de clases Java para el desarrollo de interfaces de usuario gráficas (39).

Dentro de los componentes que engloba AWT se utilizó para la implementación (39):

**Swing:** es la parte más importante y la que más desarrollada se encuentra. Ha sido creada de conjunción con Netscape y proporciona una serie de componentes muy bien descritos y especificados de forma que su presentación visual es independiente de la plataforma en que se ejecute el applet o la aplicación que utilice estas clases. **Swing** simplemente extiende el AWT añadiendo un conjunto de componentes, *JComponents* y sus clases de soporte. También proporciona otros widgets nuevos como árboles y pestañas.

### 1.9 Pruebas de Caja Blanca

La prueba de Caja Blanca del software se basa en un examen cercano al detalle procedimental. Se prueban las rutas lógicas del *software* y la colaboración entre componentes, al proporcionar casos de pruebas que ejerciten conjuntos específicos de condiciones, bucles o ambos (24).

Requieren del conocimiento de la estructura interna del programa, deben garantizar como mínimo que (40):

- Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas.

- Se ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Se ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los métodos de pruebas más importantes que se le aplican a los productos software, logrando como resultado identificar en un gran porcentaje el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad (40).

## 1.10 Pruebas de Caja Negra

Permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos (24).

Las pruebas de caja negra pretenden encontrar estos tipos de errores (24):

- Funciones incorrectas o ausentes
- Errores en la interfaz
- Errores de rendimiento
- Errores de inicialización y de terminación

## 1.11 Conclusiones del capítulo

Al concluir el capítulo quedaron definidas las bases para el inicio del desarrollo de la aplicación. Luego de realizado un estudio sobre los elementos fundamentales de composición y realización del mapeo relacional de objetos, además de una descripción de las principales propiedades que posee Doctrine fundamentalmente en su versión 2, se determinó realizar una aplicación de escritorio que abarque las principales ventajas e incluya solución a las desventajas de las ya existentes, así como que cubra la mayor parte de las necesidades de los equipos de desarrollo que utilizan dicho framework de persistencia en CEGEL. Se definió que la metodología de desarrollo ágil XP rigiera el proceso de desarrollo de software, se utilizó el lenguaje de programación Java, el IDE empleado fue Netbeans 8.0, integrado con la librería AWT. Además, se expusieron los métodos de pruebas de Caja Blanca y Caja Negra que se realizaron al software.

## Capítulo 2. Diseño e implementación

### 2.1 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. Para una mayor documentación de respaldo a la herramienta desarrollada, son elaborados los artefactos necesarios correspondientes a la metodología de desarrollo de software seleccionada. Igualmente, se especifican los patrones de diseño y la arquitectura utilizada en la concepción de la solución.

### 2.2 Descripción de la solución

La solución propuesta que lleva por nombre ORM MAPPING, es una aplicación de escritorio de código abierto 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 para brindar 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.

Con el desarrollo de la herramienta se pretende dar solución a los problemas que tienen los equipos de desarrollo, a la hora de generar los ficheros de mapeo de los esquemas persistentes, brindándole la posibilidad al desarrollador de configurar según lo que establece la arquitectura del proyecto, la forma en que la aplicación generará las salidas para los ficheros de mapeo. Para complementar lo antes expuesto, el sistema permite personalizar el mapeo en cuanto a relaciones que existen entre los objetos presentes en el mismo, relaciones como las de uno a uno, uno a mucho, mucho a mucho y herencia. Teniendo en cuenta que en el centro se desarrollan las soluciones empleando el lenguaje PHP con PostgreSQL como SGBD, la herramienta permite conectividad con este gestor, facilitando el mapeo de las clases existentes y generando los correspondientes ficheros.

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 antes mencionado, 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. 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. ORM MAPPING permite que todo el trabajo se realice a través de una interfaz visual fácil de manipular por los arquitectos de datos.

### Actores del sistema

A continuación se muestran las personas relacionadas con el sistema, en este caso los arquitectos de datos.

Tabla 5. Personas relacionadas con el sistema

Persona	Descripción
Arquitectos de datos	Persona que se encarga de identificar los datos que se almacenarán en la base de datos y elegir las estructuras apropiadas para la misma.

### 2.3 Especificación de los requisitos del software

La ingeniería de requisitos es la etapa más crucial del desarrollo de un proyecto de software. Los requisitos son el conjunto de técnicas y procedimientos que permiten conocer los elementos necesarios para definir un proyecto de software, es en la ingeniería de requisitos, donde se lleva a cabo el proceso de descubrir, analizar, escribir y verificar los servicios y restricciones del sistema que se desean producir; este proceso se realiza mediante la obtención, el análisis, la especificación, la validación y la administración de los requisitos del software. Los requisitos de software son las necesidades de los clientes, los servicios que los usuarios desean que proporcione el sistema y las restricciones en las que debe operar (41).

#### 2.3.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir (24). Para la construcción del software se definieron los requisitos funcionales siguientes:

- **RF1: Adicionar entidad:** permite adicionar entidades al diagrama de clases persistentes.
- **RF2: Modificar entidad:** modifica una entidad disponible en el diagrama de clases persistentes.
- **RF3: Eliminar entidad:** elimina una entidad del diagrama de clases persistentes.

- **RF4: Buscar entidad:** permite buscar una entidad dentro del diagrama de clases persistentes.
- **RF5: Adicionar atributo:** permite adicionar atributos a las entidades creadas por el diseñador de bases de datos.
- **RF6: Modificar atributo:** modifica un atributo disponible en la entidad.
- **RF7: Eliminar atributo:** elimina un atributo de la entidad.
- **RF8: Listar atributos:** muestra una lista de todos los atributos de la entidad.
- **RF9: Adicionar relación:** permite adicionar una relación asociada a una o varias entidades.
- **RF10: Modificar relación:** modifica el tipo de la relación.
- **RF11: Eliminar relación:** elimina las relaciones entre entidades.
- **RF12: Mostrar código:** permite visualizar el código de la entidad con las anotaciones ORM.
- **RF13: Adicionar nuevo diagrama:** crea un nuevo diagrama de clases persistentes.
- **RF14: Abrir:** permite crear un diagrama de clases persistentes a partir de un archivo con la extensión del programa.
- **RF15: Guardar:** permite guardar el diagrama de clases persistentes en un archivo con extensión propia del programa.
- **RF16: Importar ORM:** permite cargar un diagrama de clases persistentes a partir de las entidades php, las cuales contengan el mapeo relacional de objetos del marco de trabajo Doctrine.
- **RF17: Seleccionar fichero a importar:** permite buscar y seleccionar en un directorio específico el archivo que desea importar.
- **RF18: Exportar a ORM:** permite exportar un diagrama de clases persistentes como entidades php con el mapeo relacional de objetos del marco de trabajo doctrine.
- **RF19: Exportar a SQL:** permite exportar un diagrama de clases persistentes como script SQL.
- **RF20: Exportar diagrama como imagen:** permite exportar una imagen del diagrama de clases persistentes diseñado.
- **RF21: Seleccionar destino de los ficheros a guardar:** permite seleccionar la ubicación donde se quiere guardar el fichero.
- **RF22: Generar BD:** permite probar la conexión con la base de datos, crear la base de datos y crear las tablas en la base de datos a partir del diagrama de clases persistentes.
- **RF23: Probar conexión con la BD:** prueba la conexión con el gestor de bases de datos PostgreSQL.
- **RF24: Crear BD:** crea una base de datos en PostgreSQL.
- **RF25: Listar BD:** se listan todas las bases de datos disponibles en el gestor de PostgreSQL.
- **RF26: Crear esquema:** crea los esquemas en la base de datos.

- **RF27: Sincronizar ORM:** permite la actualización de las clases php luego de haberlas importado o exportado.
- **RF28: Sincronizar BD:** permite la actualización de las tablas en la base de datos luego de haberlas creado.

### 2.3.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no incluyen una relación directa con el comportamiento funcional del sistema, sino con las propiedades que este debe tener. Describen aspectos del software que generalmente pueden ser visibles por el usuario (42). Para la confección de la aplicación se identificaron los siguientes requisitos no funcionales:

#### 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 clases o de diseño de base de datos.

#### Requerimientos de portabilidad:

- **RNF8:** la herramienta desarrollada deberá ser multiplataforma teniendo un correcto funcionamiento tanto en Linux como en Windows.

#### Requerimientos de eficiencia:

- **RNF9:** tiempo de respuesta: la mayoría de los procesos que se implementan con transacciones donde se modifica la base de datos deben tener tiempos de respuesta no mayores a los 3 segundos. En el caso de la obtención de información desde las entidades, se busca que el tiempo de respuesta se simplifique al máximo y no exceda los 20 segundos.

#### Requerimientos de software:

- **RNF10:** 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.x y como IDE Netbeans 8.0.

- **RNF11:** sistema operativo Microsoft Windows 7 o superior o cualquier distribución de Linux.

### Requerimientos del diseño y de implementación:

- **RNF12:** el sistema implementado será una aplicación de escritorio.
- **RNF13:** el sistema será diseñado siguiendo los principios de programación orientada a objeto.
- **RNF14:** el sistema se implementará usando NetBeans 8.0.
- **RNF15:** el lenguaje de programación a utilizar será Java.

### Requerimientos de funcionalidad:

- **RNF16:** el sistema mostrará los errores en forma de mensajes. Todos los mensajes de error del sistema deberán incluir una descripción textual del mismo.

### 2.3.3 Validación de requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran correctos y cumplieran con las necesidades del cliente. Se tuvieron en cuenta las métricas para medir la estabilidad y especificidad.

#### Aplicación de la métrica Estabilidad de requisitos:

Es necesario lograr una estabilidad en los requisitos para el correcto funcionamiento de los demás flujos de trabajo. Se considera que los requisitos son estables cuando no existen adiciones o supresiones en ellos que impliquen modificaciones en las funcionalidades principales de la aplicación (43).

$$ETR = \left[ \frac{RT - RM}{RT} \right] \times 100$$

Donde:

1. ETR: valor de la estabilidad de los requisitos.
2. RT: total de requisitos definidos.
3. RM: número de requisitos modificados, que se obtienen como la sumatoria de los requisitos insertados, modificados y eliminados.

Esta métrica ofrece valores entre 0 y 100. El mejor valor de ETR es el más cercano a 100. Teniendo en cuenta que se identificaron un total de 28 requisitos funcionales, de los cuales 3 resultaron modificados, se calcula:

$$ETR = [(28 - 3) / 28] * 100 = 89,28$$

Como resultado se obtuvo un valor de 89.28. Dicha cifra demuestra que no se han realizado cambios significativos sobre los requisitos, son estables y, por tanto, es confiable trabajar el diseño sobre ellos.

### **Aplicación de la métrica Especificidad de los requisitos:**

El objetivo de esta métrica es cuantificar la especificidad o falta de ambigüedad en la definición de los requisitos. Para calcular esta métrica se contaron los requisitos que tuvieron igual interpretación por los revisores y se compararon con el total de requisitos definidos (43).

La especificidad de los requisitos se calcula como:

$$ER = n_{ui}/n_r$$

Donde:

- ✓ ER: grado de especificidad de los requisitos.
- ✓  $n_{ui}$ : número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.
- ✓  $n_r$ : cantidad de requisitos en una especificación.

El valor de esta métrica debe estar siempre entre 0 y 1. Mientras más cerca de 1 esté el valor de ER mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos.

Luego:

$$ER = n_{ui} / n_r = 28 / 28 = 1$$

Como resultado se obtuvo que la especificación de los requisitos no presenta ambigüedad, asegurando un alto nivel de calidad en el proceso de especificación.

### **2.4 Descripción de la arquitectura**

El desarrollo de la solución presenta una arquitectura en capas, pues esta facilita tanto la comprensión como la construcción del sistema, se seleccionó teniendo en cuenta las características particulares de la herramienta a desarrollar en cuanto a la lógica del negocio. La organización de las capas en la aplicación quedaría agrupada en tres niveles, compuesta por la capa Presentación, Negocio y Acceso a Datos. En la figura 2 se muestra la vista de la arquitectura de la aplicación.

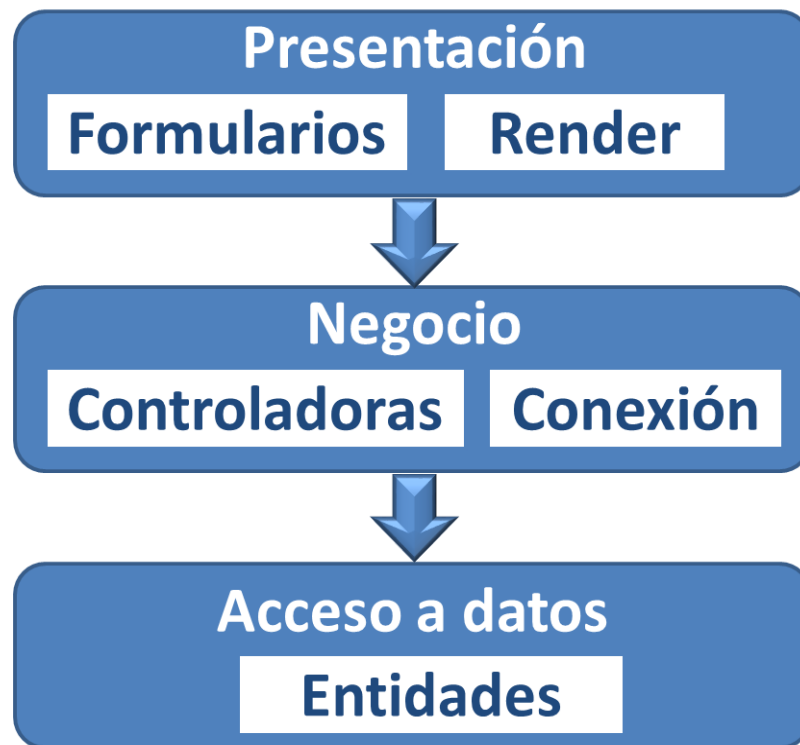


Figura 2. Arquitectura por capas. (Elaboración propia)

La arquitectura tiene como objetivo principal separar los diferentes aspectos del desarrollo en que está conformado el sistema, minimizando las dependencias entre capas. A continuación se describen los tres niveles antes mencionados (44).

### **Nivel de Presentación**

Es la capa que contiene los componentes con los que interactuará el usuario. Esta tiene acceso a las funcionalidades que brinda el negocio y puede mostrar o capturar la información a través de los diferentes elementos que incluye (44). En este nivel se ubican los formularios de interfaz de usuario, los cuales responden a un diseño y comportamiento estándar con vista a facilitar el trabajo con ellos y lograr la estandarización de la aplicación, además se encuentran las clases encargadas del modelado del diagrama de clases persistentes. En esta capa el usuario trabaja directamente con la información de las entidades de forma visual.

### **Nivel de Negocio**

Esta capa se encarga de recibir una petición de la capa superior, gestionar o procesar la misma, de ser necesario solicitándola a la capa de Acceso a Datos y finalmente envía la respuesta a quien realizó la petición (44). Contiene clases encargadas de resolver determinadas funcionalidades y clases diseñadas para realizar pruebas de unidad u otras a los diferentes bloques de código de este nivel. Además, encierra las clases que implementan la totalidad de las operaciones de persistencia

y obtención de datos, así como la estructura que presenta la capa para el trabajo correcto de determinadas funcionalidades para la persistencia.

### Nivel de Acceso a Datos

Es la capa inferior, por lo que los componentes que contiene desconocen los niveles superiores. Este nivel se limita al manejo de la información, ya sea para persistirla u obtenerla para su procesamiento y propagación por la aplicación (44).

## 2.5 Fases del proceso de desarrollo

### 2.5.1 Fase de exploración

En esta fase los clientes describen las HU definiendo las características que van a tener cada una de ellas.

### HU

Las HU son realizadas por el cliente, el cual en su propio lenguaje describe lo que el sistema debe cumplir. Las mismas deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llega el momento de la implementación, los desarrolladores se reunirán directamente con el cliente para obtener toda una lista de detalles necesarios para satisfacer sus necesidades (45). A continuación se describe la estructura de algunas HU que poseen prioridad alta, las restantes se encuentran en el documento oficial que se elaboró *HU\_Aplicación Informática para realizar el mapeo relacional de clases de Doctrine*, además, se muestran en los anexos.

Tabla 6. HU Exportar a ORM

Historia de usuario	
<b>Número:</b> 18	<b>Nombre:</b> Exportar a ORM.
<b>Cantidad de modificaciones a la Historia de Usuario:</b> 0	
<b>Usuario:</b> Diseñador de bases de datos.	<b>Iteración asignada:</b> 2
<b>Prioridad en negocio:</b> alta	<b>Puntos estimados:</b> 3
<b>Riesgo en desarrollo:</b> medio	<b>Puntos reales:</b> 2
<b>Descripción:</b> permite exportar un diagrama de clases persistentes como entidades php con el mapeo relacional de objetos del marco de trabajo Doctrine.	

**Observaciones:** para exportar las entidades php es necesario realizar un diagrama de clases persistentes.

**Prototipo de interfaz:**

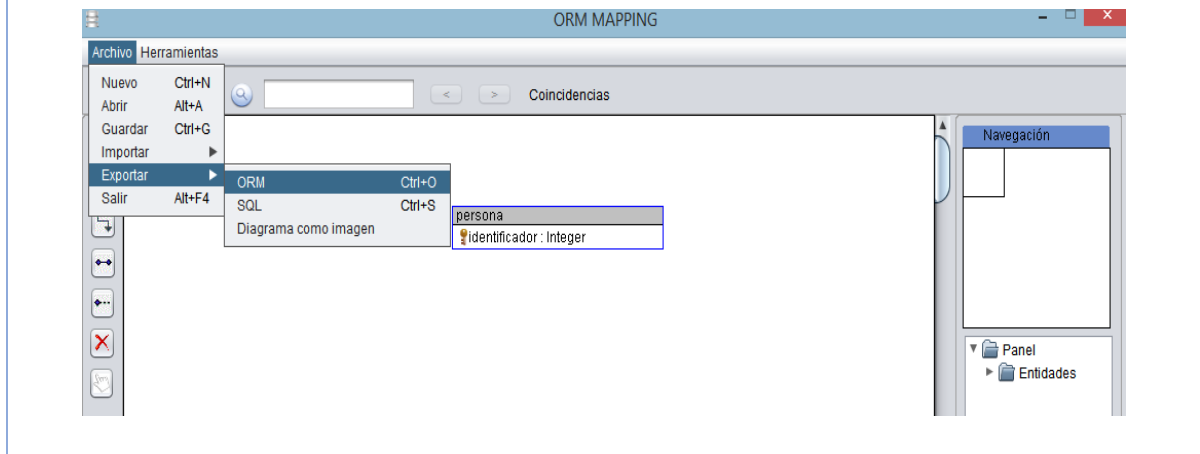


Tabla 7. HU Importar ORM

Historia de Usuario	
<b>Número:</b> 16	<b>Nombre de la Historia de Usuario:</b> Importar ORM.
<b>Cantidad de modificaciones a la Historia de Usuario:</b> 0	
<b>Usuario:</b> Diseñador de bases de datos.	<b>Iteración asignada:</b> 3
<b>Prioridad en negocio:</b> alta	<b>Puntos estimados:</b> 3
<b>Riesgo en desarrollo:</b> medio	<b>Puntos reales:</b> 3
<b>Descripción:</b> permite crear un diagrama de clases persistentes a partir de las entidades php, las cuales contengan el mapeo relacional de objetos del marco de trabajo doctrine.	
<b>Observaciones:</b> solo se pueden importar entidades php.	



## Prototipo de interfaz

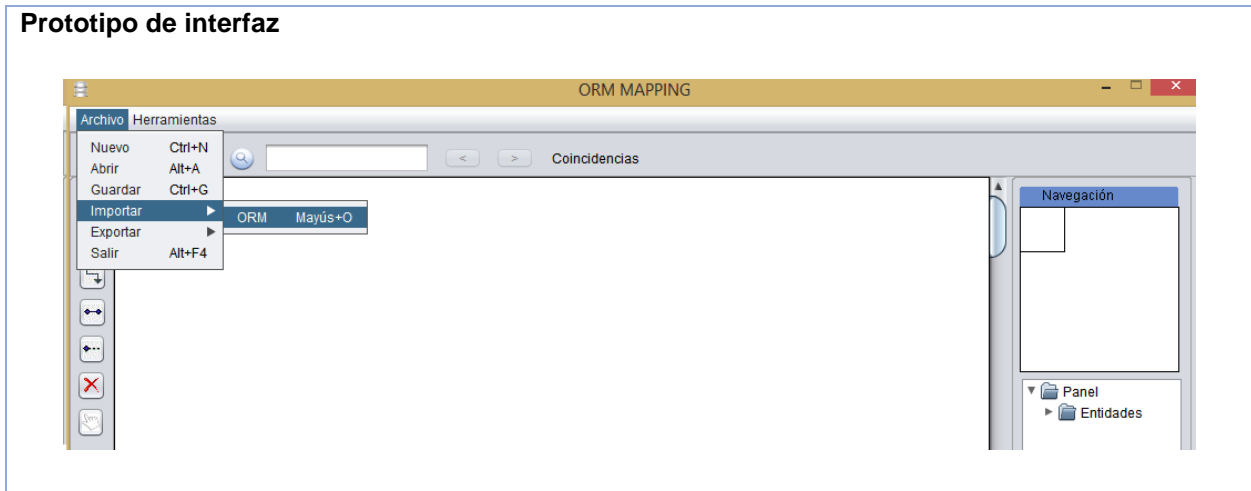
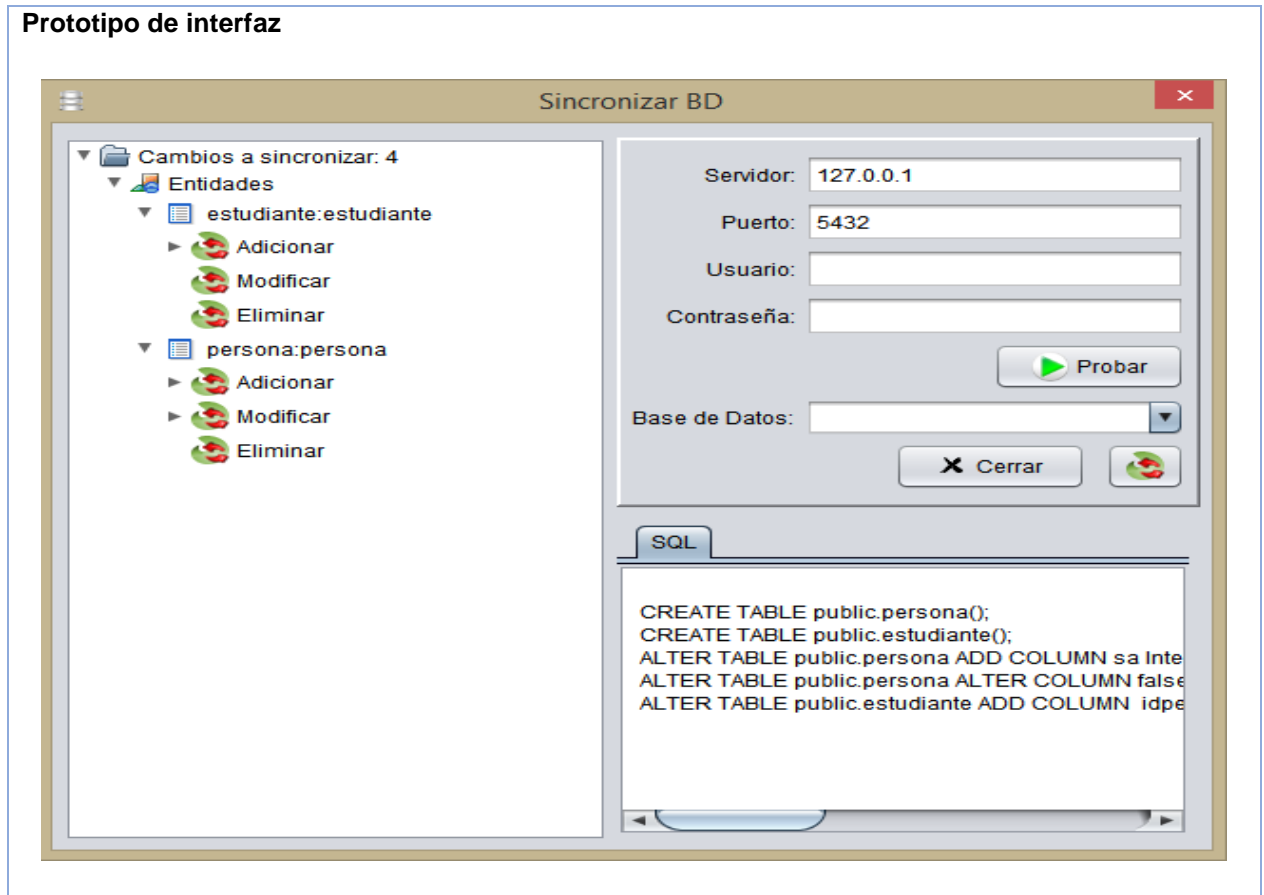


Tabla 8. HU Sincronizar BD

Historia de Usuario	
<b>Número:</b> 28	<b>Nombre de la Historia de Usuario:</b> Sincronizar BD.
<b>Cantidad de modificaciones a la Historia de Usuario:</b> 0	
<b>Usuario:</b> Diseñador de bases de datos.	<b>Iteración asignada:</b> 3
<b>Prioridad en negocio:</b> alta	<b>Puntos estimados:</b> 3
<b>Riesgo en desarrollo:</b> medio	<b>Puntos reales:</b> 3
<b>Descripción:</b> permite la actualización de las tablas en la base de datos luego de haberlas creado.	
<b>Observaciones:</b> debe elegir la misma base de datos con la que fue generada inicialmente.	

### Prototipo de interfaz



A continuación se explica cada uno de los datos que deben ser llenados en la HU:

- **Número:** identificador de la HU.
- **Nombre:** nombre que identifica a la HU.
- **Usuario:** involucrados en la ejecución de la HU.
- **Iteración asignada:** iteración en que se implementará la HU.
- **Prioridad en el negocio:** prioridad de la HU respecto al resto de las HU, puede ser: alta, media o baja.
- **Riesgo en desarrollo:** riesgo en la implementación de la HU, puede ser: alto, medio o bajo.
- **Puntos estimados:** estima el esfuerzo asociado a la implementación de la HU. Un punto equivale a una semana ideal de programación, generalmente de 1 a 3 puntos.
- **Puntos reales:** resultado del esfuerzo asociado a la implementación de la HU. Un punto equivale a una semana ideal de programación, generalmente de 1 a 3 puntos.
- **Descripción:** descripción sintetizada de la HU.
- **Observaciones:** información de interés.
- **Prototipo de interfaz:** imagen de la funcionalidad desarrollada.

### 2.5.2 Fase de planificación

En la fase de planificación se realiza la estimación del esfuerzo que causará la implementación de cada HU, como en la metodología ágil XP las métricas son libres, puede utilizarse cualquier criterio definido para medir el desempeño del proyecto en cuestión (22).

#### Estimación de esfuerzo por HU

Para la realización de la aplicación propuesta se efectuó una estimación de esfuerzo por cada una de las HU identificadas, donde a continuación se muestran los resultados. Los puntos de estimación están en un rango de 1 a 3.

Tabla 9. Estimación del esfuerzo por HU

Historia de usuario	Puntos de estimación (semanas)
Adicionar entidad	1
Modificar entidad	1
Eliminar entidad	1
Buscar entidad	1
Adicionar atributo	1
Modificar atributo	1
Eliminar atributo	1
Listar atributos	1
Adicionar relación	1
Modificar relación	1
Eliminar relación	1
Mostrar código	1
Adicionar nuevo diagrama	1
Abrir	1
Guardar	1
Importar ORM	3

Seleccionar fichero a importar	2
Exportar a ORM	3
Exportar a SQL	3
Exportar diagrama como imagen	2
Seleccionar destino de los ficheros a guardar	1
Generar BD	2
Probar conexión con la BD	1
Crear BD	1
Listar BD	1
Crear esquema	1
Sincronizar ORM	3
Sincronizar BD	3

### Plan de iteraciones

Luego de haber identificado las HU y de realizar una estimación del tiempo requerido para la realización de cada una, se realizó la planificación estableciendo las iteraciones que necesarias antes de entregar el sistema.

- Iteración 1: tiene como objetivo la implementación de las HU: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14 y 15.
- Iteración 2: tiene como objetivo la implementación de las HU: 12, 18, 19, 20, 21.
- Iteración 3: tiene como objetivo la implementación de las HU: 16, 22, 23, 24, 25, 26, 27 y 28.

### Plan de duración de las iteraciones

En el plan de duración de las iteraciones se muestran las HU que son implementadas en cada una de las iteraciones, así como la duración estimada y el orden de implementación de cada una de ellas (45), (ver tabla 10).

Tabla 10. Plan de duración de iteraciones

Iteración	Orden de las Historias de usuario a implementar	Semanas
1	<ul style="list-style-type: none"> <li>➤ Adicionar entidad</li> <li>➤ Modificar entidad</li> </ul>	1

	<ul style="list-style-type: none"> <li>➤ Eliminar entidad</li> <li>➤ Buscar entidad</li> <li>➤ Adicionar atributo</li> <li>➤ Modificar atributo</li> <li>➤ Eliminar atributo</li> <li>➤ Listar atributos</li> <li>➤ Adicionar relación</li> <li>➤ Modificar relación</li> <li>➤ Eliminar relación</li> <li>➤ Adicionar nuevo diagrama</li> <li>➤ Abrir</li> <li>➤ Guardar</li> </ul>	
2	<ul style="list-style-type: none"> <li>➤ Exportar ORM</li> <li>➤ Exportar SQL</li> <li>➤ Mostrar código</li> <li>➤ Exportar diagrama como imagen</li> <li>➤ Seleccionar destino de los ficheros a guardar</li> </ul>	4
3	<ul style="list-style-type: none"> <li>➤ Generar BD</li> <li>➤ Importar ORM</li> <li>➤ Sincronizar ORM</li> <li>➤ Sincronizar BD</li> <li>➤ Seleccionar fichero a importar</li> <li>➤ Probar conexión con la BD</li> <li>➤ Crear BD</li> <li>➤ Listar BD</li> <li>➤ Crear esquema</li> </ul>	5

### Plan de entrega

El plan de entrega es el compromiso final del equipo de desarrollo con los clientes. Es de vital importancia ya que la entrega tardía de la solución repercute negativamente en el desarrollo del producto, creando insatisfacción en el cliente (45). En esta etapa se definió para cada iteración una fecha de inicio y fin (ver tabla 11).

Tabla 11. Plan de entregas

Iteración	Historia de usuario	Estimación (semana)	Fecha
1	<ul style="list-style-type: none"> <li>➤ Adicionar entidad</li> <li>➤ Modificar entidad</li> <li>➤ Eliminar entidad</li> </ul>	1	16/2/2015-23/2/2015

	<ul style="list-style-type: none"> <li>➤ Buscar entidad</li> <li>➤ Adicionar atributo</li> <li>➤ Modificar atributo</li> <li>➤ Eliminar atributo</li> <li>➤ Listar atributos</li> <li>➤ Adicionar relación</li> <li>➤ Modificar relación</li> <li>➤ Eliminar relación</li> <li>➤ Adicionar nuevo diagrama</li> <li>➤ Abrir</li> <li>➤ Guardar</li> </ul>		
2	<ul style="list-style-type: none"> <li>➤ Exportar a ORM</li> <li>➤ Exportar a SQL</li> <li>➤ Mostrar código</li> <li>➤ Exportar diagrama como imagen</li> <li>➤ Seleccionar destino de los ficheros a guardar</li> </ul>	4	24/2/2015- 24/3/2015
3	<ul style="list-style-type: none"> <li>➤ Generar BD</li> <li>➤ Importar ORM</li> <li>➤ Sincronizar ORM</li> <li>➤ Sincronizar BD</li> <li>➤ Seleccionar fichero a importar</li> <li>➤ Probar conexión con la BD</li> <li>➤ Crear BD</li> <li>➤ Listar BD</li> <li>➤ Crear esquema</li> </ul>	5	25/3/2015- 30/4/2015

## 2.6 Fase de diseño

### 2.6.1 Patrones de diseño

El conocimiento de los patrones de diseño es vital para entender el funcionamiento del mismo y por ende lograr los objetivos trazados. Por esta razón uno de los pasos a tener en cuenta es identificar qué patrones pueden ser utilizados. A continuación se muestran los patrones aplicados en la confección de la herramienta.

### Patrones para Asignar Responsabilidades (GRASP)

Durante el diseño de la herramienta se emplearon patrones GRASP, específicamente:

**Experto:** el patrón experto es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo (26). Se evidencia por ejemplo en la clase “Sistema”, pues cuenta con la información necesaria para cumplir responsabilidades específicas.

**Creador:** el patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases (26). Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador, este patrón se pone de manifiesto en las clases: Sistema, Exportar y GenerarBd. A la hora de crear objetos se deben tener en cuenta las características de la clase. A continuación se muestra un ejemplo de su uso en la aplicación.

```
TipoRelacion TR = new TipoRelacion("*", "*",...)
```

Figura 3. Ejemplo de uso del patrón Creador. (Elaboración propia)

**Controlador:** el patrón controlador es un patrón que sirve como intermediario entre una interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según corresponda (26). La clase “Sistema” es la responsable de atender los eventos del sistema y además, encargada de capturar las llamadas que se efectúan en la aplicación en tiempo real.

**Alta Cohesión:** la cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento. Se evidencia en cada clase del diseño propuesto que realiza una labor única dentro del sistema y colabore con las otras para llevar a cabo una tarea, como son las clases que forman parte de las capas negocio y acceso a datos (26). A continuación se muestra un ejemplo de su uso.

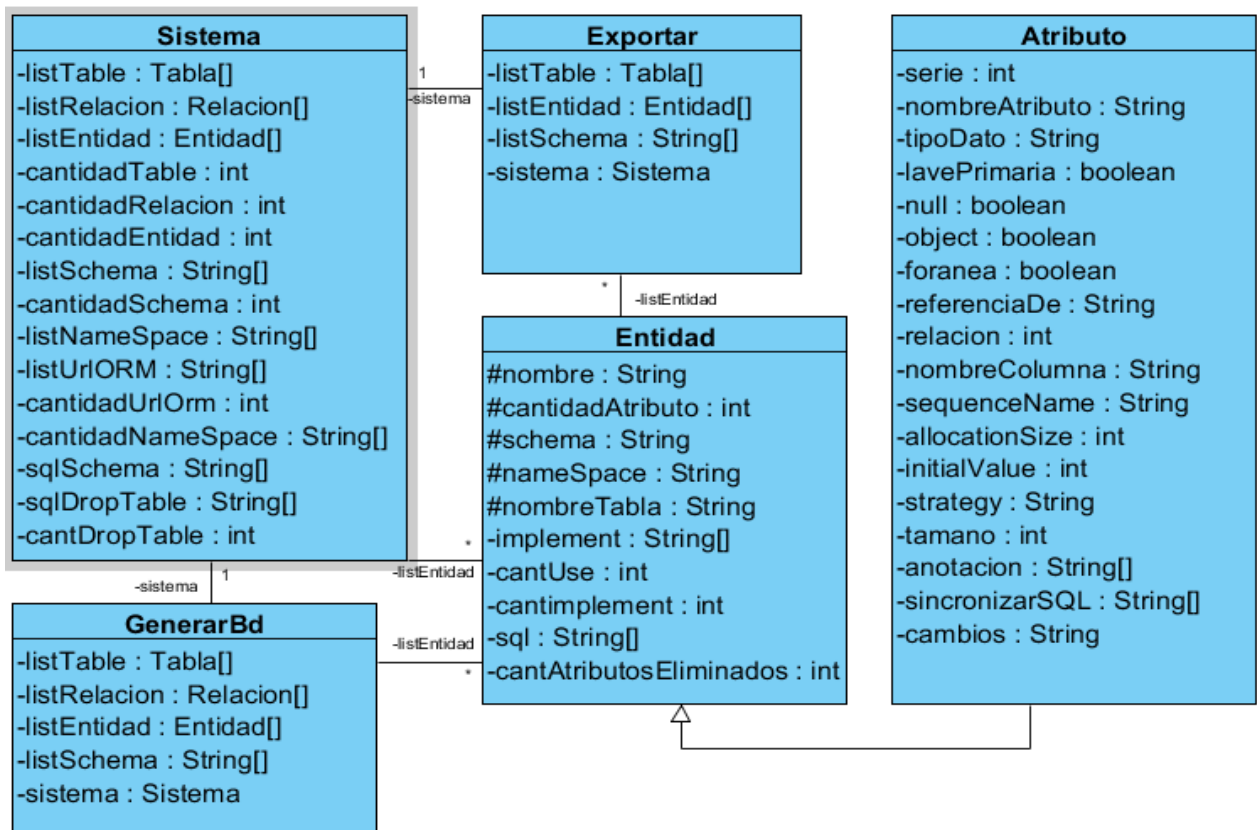


Figura 4. Ejemplo de uso del patrón Alta Cohesión. (Elaboración propia)

**Bajo Acoplamiento:** se pone en práctica en las clases que poseen pocas relaciones entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás. Esta característica permitió potenciar la reutilización y disminuyó la dependencia entre las clases (26).

## Patrones GoF (Gang of Four)

En el diseño fueron tomados en cuenta los siguientes patrones GOF:

### Creacionales:

- **Fábrica abstracta:** el problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos (ventana, menú, botón) (26). Se pone de manifiesto en las clases: AdicionarTabla, Cargando, CodigoTabla, Principal, SincronizarBd, Conectar. En la figura 7 se muestra un ejemplo de su utilización.



```
private void initComponents() {
    jMenuBar2 = new javax.swing.JMenuBar();
    jMenu3 = new javax.swing.JMenu();
    jMenu6 = new javax.swing.JMenu();
    jMenu8 = new javax.swing.JMenu();
    jPopupMenu1 = new javax.swing.JPopupMenu();
    menuBar1 = new java.awt.MenuBar();
}
```

Figura 5. Ejemplo de uso del patrón Fábrica Abstracta. (Elaboración propia)

## 2.6.2 Tarjetas de Clase, Responsabilidad y Colaboración (CRC)

Para el diseño de las aplicaciones, la metodología XP no requiere la representación del sistema mediante diagramas de clases utilizando notación UML, en su lugar se usan otras técnicas como las tarjetas CRC. La técnica CRC propone una forma de trabajo, preferentemente grupal, para encontrar los objetos del dominio de la aplicación, sus responsabilidades y cómo colaboran con otros para realizar tareas (46).

Solo se realizaron las tarjetas para las clases controladoras del sistema, ya que estas son las que controlan principalmente el flujo de información. A continuación se muestra la tarjeta CRC de una de las clases controladoras, las restantes se encuentran disponibles en los anexos.

Tabla 12. Descripción Tarjeta CRC Sistema

Tarjeta CRC	
<b>Clase:</b> Sistema	
<b>Descripción:</b> define las funcionalidades que deben ser implementadas para realizar proceso de mapeo relacional de objetos.	
Responsabilidades	Colaboraciones
<ul style="list-style-type: none"> <li>➤ Crear el diagrama de clases persistentes.</li> <li>➤ Adicionar entidad</li> <li>➤ Adicionar atributo</li> <li>➤ Adicionar relación</li> <li>➤ Adicionar esquema</li> <li>➤ Adicionar nameSpace</li> </ul>	<ul style="list-style-type: none"> <li>➤ Entidad</li> <li>➤ Relacion</li> <li>➤ Ptabla</li> </ul>

## 2.6.3 Diagrama de clases del diseño

Un diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. El mismo representa las

clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Además, sirve para visualizar las relaciones entre las clases que involucran el sistema (47).

A pesar de que la metodología escogida no define diagramas en notación UML, dentro de sus artefactos, se decidió mostrar el diagrama de clases del diseño para un mejor entendimiento de la estructura y las relaciones entre clases de la solución propuesta, esta cuenta con los siguientes elementos que se pueden visualizar en la figura 6.

- Clase: atributos y visibilidad
- Relaciones: herencia y uso

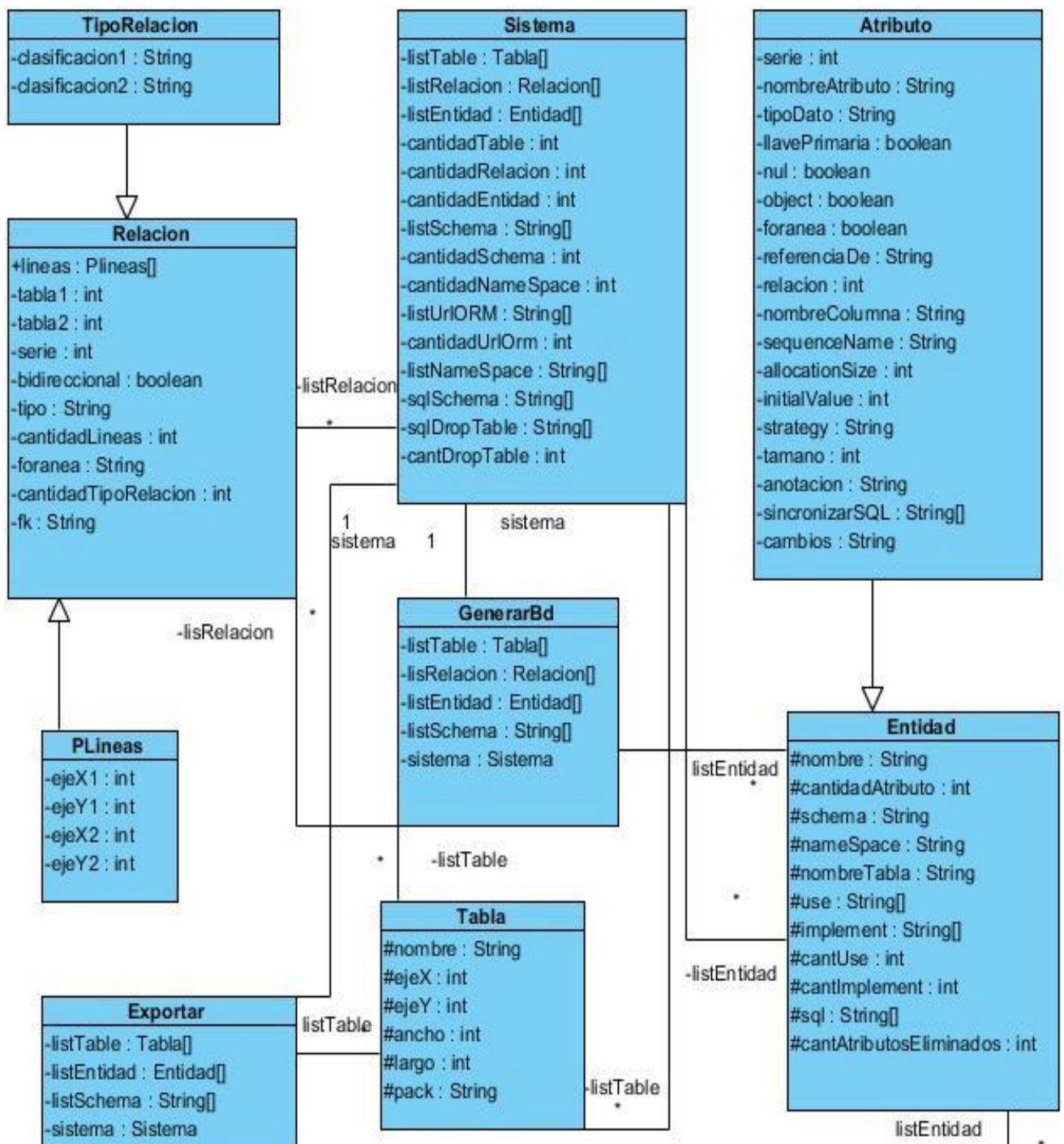


Figura 6. Diagrama de clases del diseño. (Elaboración propia)

A continuación se describen brevemente las clases presentadas en el diagrama de la figura 6.

Tabla 13. Descripción de las clases del diagrama de diseño

Clases	Descripción
Sistema	Es la clase que se encarga de la creación del modelo del diagrama de clases persistentes, así como almacenar la información de las clases de acceso a datos.
GenerarBd	Es la clase que se encarga de la generación de la base de datos con sus respectivas tablas y el proceso de sincronización.
Exportar	Es la clase que contiene las funcionalidades que permiten exportar el diagrama de clases persistentes a entidades php y el script sql.
Entidad	Es la clase que contiene la información de las entidades.
Relacion	Es la clase que contiene la información acerca de las relaciones entre entidades.
PLineas	Es la clase encargada de dibujar las relaciones entre las entidades.
TipoRelacion	Clase que contiene la clasificación de las relaciones.
Tabla	Clase que dibuja el estereotipo de la entidad.
Atributo	Es la clase que contiene la información de los atributos pertenecientes a cada entidad.

## 2.6.4 Validación del diseño

Un aspecto importante a tener en cuenta en la evaluación del diseño, es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Para medir la calidad del diseño realizado se tuvieron en cuenta distintos atributos como son la responsabilidad de las clases, la reutilización y la complejidad de implementación y mantenimiento. Las métricas Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC) permiten evaluar estos atributos y adicionalmente el bajo acoplamiento y la cantidad de pruebas unitarias que deben realizarse (48).

### Métrica tamaño operacional de clase (TOC)

A continuación se muestra un ejemplo del resultado de la aplicación de esta métrica al diseño.

Tabla 14. Responsabilidad

Responsabilidad	Cantidad de clases	Promedio
Baja	9	69.23076923
Media	2	15.38461538
Alta	2	15.38461538



Figura 7. Representación de la evaluación de la métrica TOC para el atributo responsabilidad. (Elaboración propia)

Tabla 15. Complejidad

Complejidad	Cantidad de clases	Promedio
Baja	9	69.23076923
Media	2	15.38461538
Alta	2	15.38461538

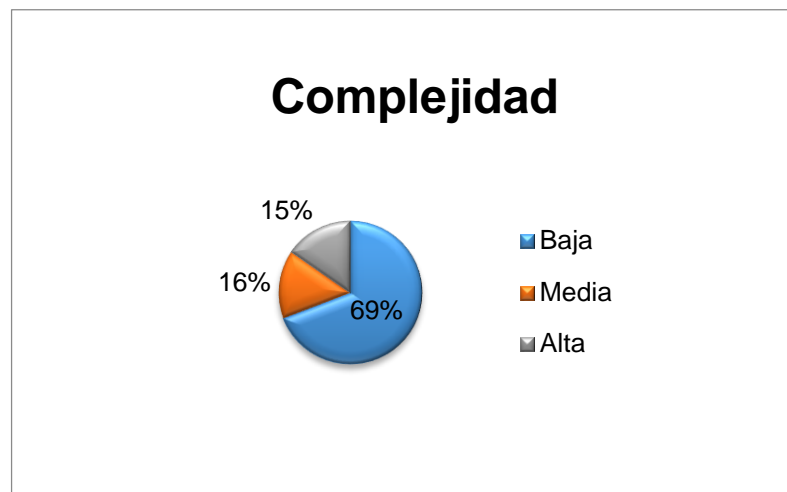


Figura 8. Representación de la evaluación de la métrica TOC para el atributo complejidad. (Elaboración propia)

Tabla 16. Reutilización

Reutilización	Cantidad de clases	Promedio
Alta	9	69.23076923
Media	2	15.38461538
Baja	2	15.38461538



Figura 9. Representación de la evaluación de la métrica TOC para el atributo reutilización. (Elaboración propia)

Después de aplicar la métrica TOC a un total de 13 clases, se llega a la conclusión de que el diseño propuesto cumple con lo requerido inicialmente ya que el mismo arrojó como resultado una alta reutilización, baja responsabilidad y complejidad, permitiendo la eliminación de clases repetidas, teniendo en cuenta que más de la mitad de las clases poseen baja dependencia respecto a otras, evidenciando que el sistema no presenta una compleja implementación.

### Relaciones entre clases (RC)

A continuación se muestra un ejemplo del resultado de la aplicación de esta métrica al diseño.

### Acoplamiento

Tabla 17. Acoplamiento

Acoplamiento	Cantidad de clases	Promedio
Ninguno	1	7.692307692
Bajo	7	53.84615385
Medio	0	0
Alto	5	38.46153846

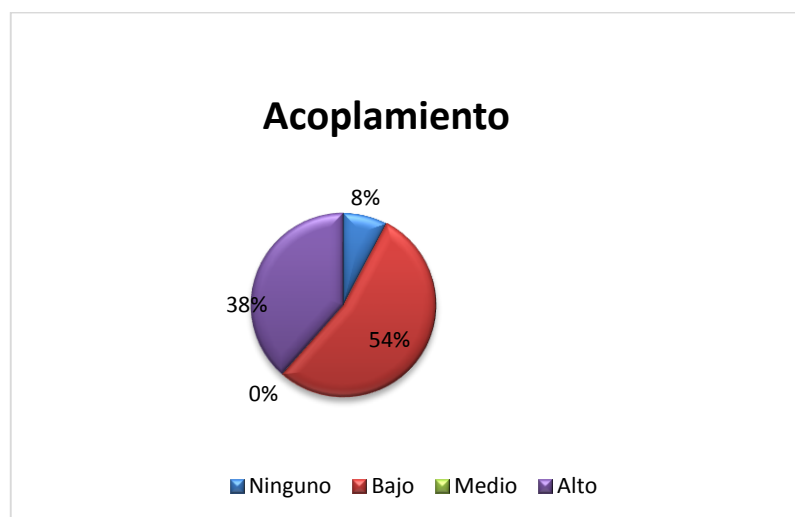


Figura 10. Representación de la evaluación de la métrica RC en el atributo Acoplamiento. (Elaboración propia)

Tabla 18. Complejidad de Mantenimiento

Complejidad de mantenimiento	Cantidad de clases	Promedio
Baja	8	61.53846154
Media	5	38.46153846
Alta	0	0



Figura 11. Representación de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. (Elaboración propia)

Tabla 19. Cantidad de pruebas

Cantidad de pruebas	Cantidad de clases	Promedio
Baja	8	61.53846154
Media	5	38.46153846
Alta	0	0

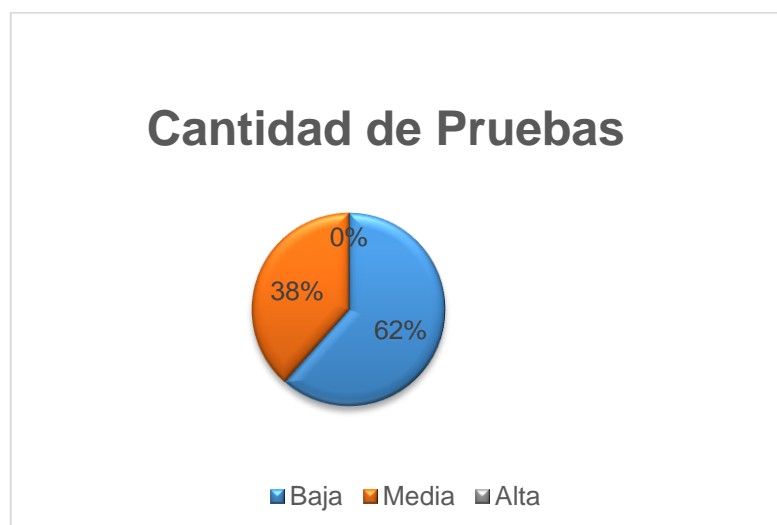


Figura 12. Representación de la evaluación de la métrica RC en el atributo cantidad de pruebas. (Elaboración propia)

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que el diseño propuesto se encuentra dentro de los niveles de calidad requeridos. Los atributos de calidad fueron evaluados satisfactoriamente confirmando que las clases están diseñadas correctamente ya que presentan un bajo acoplamiento, poca complejidad de mantenimiento y poca cantidad de pruebas que implica que se necesita menos esfuerzo a la hora de realizar pruebas unitarias a estas clases.



## 2.7 Fase de implementación

La implementación en XP tiene varias características particulares de la propia metodología. Las tareas de ingeniería generadas por las HU se desarrollaron siguiendo las siguientes prácticas (49):

- Adoptar un método de desarrollo basado en pruebas para asegurar que el código se comporta según lo esperado.
- Programación por parejas, para incrementar el conocimiento, la experiencia y las ideas.
- Asumir la propiedad colectiva del código, para que todo el equipo sea responsable de él.
- Integración continua, para reducir el impacto de la incorporación de nuevas funcionalidades.

### 2.7.1 Tareas de Ingeniería

Las tareas de ingeniería se usan para desglosar en actividades las HU. En ellas se especifica la fecha de inicio y fin de cada una, se nombra al programador responsable de cumplirla y se describe qué se deberá hacer en la misma (50). A continuación se detallan las tareas de ingeniería que se deben ejecutar para las HU 16 y 18 que tienen prioridad alta en el negocio, el resto se pueden encontrar en el *Documento de Tareas de Ingeniería*.

Tabla 20. Tarea de Ingeniería #1

Tarea	
<b>Número de tarea:</b> 1	<b>Nombre Historia de usuario:</b> Exportar a ORM
<b>Nombre de la tarea:</b> Realizar el estudio de ORM.	
<b>Tipo de tarea:</b> estudio	<b>Puntos Estimados(días):</b> 2
<b>Fecha inicio:</b> 22/2/2015	<b>Fecha fin:</b> 24/2/2015
<b>Programador responsable:</b> Oridalmis Ricart Rodríguez	
<b>Descripción:</b> se realiza una búsqueda sobre el mapeo relacional de objetos y se recopila la información necesaria para la realización de la funcionalidad.	

Tabla 21. Tarea de Ingeniería #2

Tarea	
<b>Número de tarea:</b> 2	<b>Nombre Historia de usuario:</b> Exportar a ORM

<b>Nombre de la tarea:</b> Implementar el método que permite buscar el directorio donde serán exportadas las entidades.	
<b>Tipo de tarea:</b> desarrollo	<b>Puntos Estimados(días):</b> 1
<b>Fecha inicio:</b> 24/2/2015	<b>Fecha fin:</b> 24/2/2015
<b>Programador responsable:</b> Oridalmis Ricart Rodríguez	
<b>Descripción:</b> se llama una interfaz que permite seleccionar el directorio donde se exportarán las entidades.	

Tabla 22.Tarea de Ingeniería #3

Tarea	
<b>Número de tarea:</b> 3	<b>Nombre Historia de usuario:</b> Exportar a ORM
<b>Nombre de la tarea:</b> Implementar el método que permite exportar a ORM.	
<b>Tipo de tarea:</b> desarrollo	<b>Puntos Estimados(días):</b> 1
<b>Fecha inicio:</b> 25/2/2015	<b>Fecha fin:</b> 25/2/2015
<b>Programador responsable:</b> Oridalmis Ricart Rodríguez	
<b>Descripción:</b> realiza una búsqueda de información en las diferentes clases y luego escribe el contenido de las entidades php.	

Tabla 23.Tarea de Ingeniería #4

Tarea	
<b>Número de tarea:</b> 4	<b>Nombre Historia de usuario:</b> Exportar a ORM
<b>Nombre de la tarea:</b> Verificar el código generado.	
<b>Tipo de tarea:</b> desarrollo	<b>Puntos Estimados(días):</b> 1
<b>Fecha inicio:</b> 3/3/2015	<b>Fecha fin:</b> 3/3/2015
<b>Programador responsable:</b> Oridalmis Ricart Rodríguez	
<b>Descripción:</b> después de la implementación de todas las funcionalidades antes descritas se procede a la verificación de las mismas a través de la prueba de caja blanca.	

Tabla 24. Tarea de Ingeniería #5

Tarea	
<b>Número de tarea:</b> 5	<b>Nombre Historia de usuario:</b> Importar ORM
<b>Nombre de la tarea:</b> Realizar el estudio de ORM.	
<b>Tipo de tarea:</b> estudio	<b>Puntos Estimados(días):</b> 2
<b>Fecha inicio:</b> 4/3/2015	<b>Fecha fin:</b> 6/3/2015
<b>Programador responsable:</b> Carlos Andrés Sarmiento Claro	
<b>Descripción:</b> se realiza una búsqueda sobre el mapeo relacional de objetos y se recopila la información necesaria para la realización de la funcionalidad.	

Tabla 25. Tarea de Ingeniería #6

Tarea	
<b>Número de tarea:</b> 6	<b>Nombre Historia de usuario:</b> Importar ORM
<b>Nombre de la tarea:</b> Implementar el método que permite Importar ORM.	
<b>Tipo de tarea:</b> desarrollo	<b>Puntos Estimados(días):</b> 1
<b>Fecha inicio:</b> 7/3/2015	<b>Fecha fin:</b> 7/3/2015
<b>Programador responsable:</b> Carlos Andrés Sarmiento Claro	
<b>Descripción:</b> lee las entidades php y a partir de un conjunto de patrones llena con información las clases.	

Tabla 26. Tarea de Ingeniería #7

Tarea	
<b>Número de tarea:</b> 7	<b>Nombre Historia de usuario:</b> Exportar a ORM
<b>Nombre de la tarea:</b> Verificar el código generado.	
<b>Tipo de tarea:</b> desarrollo	<b>Puntos Estimados(días):</b> 1
<b>Fecha inicio:</b> 8/3/2015	<b>Fecha fin:</b> 8/3/2015
<b>Programador responsable:</b> Carlos Andrés Sarmiento Claro	

**Descripción:** después de la implementación de todas las funcionalidades antes descritas se procede a la verificación de la funcionalidad a través de la prueba de caja blanca.

### 2.7.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física del código (50).

Las realizaciones de revisiones frecuentes al código, la aplicación de forma continua de técnicas y estándares de codificación definidos, junto al empleo de buenas prácticas de programación, garantizan una alta calidad en el desarrollo de la aplicación, además de proporcionar un código legible y reutilizable (50).

#### Convenciones de nombres

En la implementación del sistema los nombres se regirán por la convención del código de Java (51).

#### Clases

Los nombres de las clases deberán ser sustantivos, en el caso de ser compuestos tendrán la primera letra de cada palabra que lo forme en mayúscula. Estos deben ser simples y descriptivos. Ejemplo: `public class Sistema {}`

#### Métodos

La denominación de los métodos debe ser un infinitivo demostrando así la acción que ejecutan, cuando sean compuestos tendrán la primera letra en minúscula y la inicial de las siguientes palabras en mayúscula.

Ejemplo: `adicionarActualizarTable (int pos, String nombre, Atributo xAtributo) {}`

#### Comentarios

El uso de los comentarios hace todo el código más fácil de trabajar, sobre todo si los algoritmos largos o complejos están involucrados. Los comentarios de código de acuerdo con convenciones de Java son incluidos antes de declaraciones de variables clave, los métodos, las clases y las estructuras largas (51). En el desarrollo de la solución se utilizó el comentario de implementación de tipo `//`. Ejemplo:

```
163 //Modifica el nombre del atributo.  
164  
165 public void setNombreAtributo(String NombreAtributo) {  
166     this.NombreAtributo = NombreAtributo;  
167 }
```

Figura 13. Ejemplo de código aplicando el comentario. (Elaboración propia)

### Nomenclaturas utilizadas

**PascalCase** establece que los nombres de los identificadores, las variables, métodos y clases están compuestos por una o más palabras juntas, iniciando cada palabra con letra mayúscula. Todas las clases están nombradas siguiendo el estándar PascalCase, nombrándolas de acuerdo al propósito y la función que corresponda (50). Ejemplo:

```
public class Sistema implements Serializable {
```

Figura 14. Ejemplo de código aplicando estándares de codificación. (Elaboración propia)

**CamelCase** es similar a PascalCase con diferencia en la primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula. Esta notación se utilizó para el nombre de funciones (51).

El siguiente fragmento de código se encuentra en la clase “Sistema”, el mismo retorna una tabla dada dos coordenadas. En el código se evidencia el uso del estándar de codificación CamelCase y otras convenciones como el comentario (ver figura 15).

```
// Busca una tabla dada las coordenada x, y

public int buscarTabla(int x, int y) {
    int pos = -1;
    for (int i = 0; i < getCantidadtable(); i++) {
        int ancho = listTable[i].getAncho() + listTable[i].getEjeX();
        int alto = listTable[i].getEjeY() + listTable[i].getLargo();

        if (x >= listTable[i].getEjeX() && x <= ancho) {
            if (y >= listTable[i].getEjeY() && y <= alto + 15) {
                pos = i;
                break;
            }
        }
    }
    return pos;
}
```

Figura 15. Ejemplo de código aplicando estándares de codificación. (Elaboración propia)

### 2.8 Prototipo de interfaz de la aplicación

Un prototipo es una visión preliminar del sistema futuro, es un modelo operable, fácilmente ampliable y modificable, que tiene todas las características que hasta el momento debe tener el sistema (52).

Con el desarrollo de la aplicación se obtuvo una interfaz de usuario amigable y sencilla que permitió al usuario interactuar con el sistema sin necesitar un entrenamiento profundo. Luego de la implementación y con el uso de los estándares de codificación utilizados se logra la herramienta ORM MAPPING, la cual quedó representada a través de la siguiente interfaz gráfica:

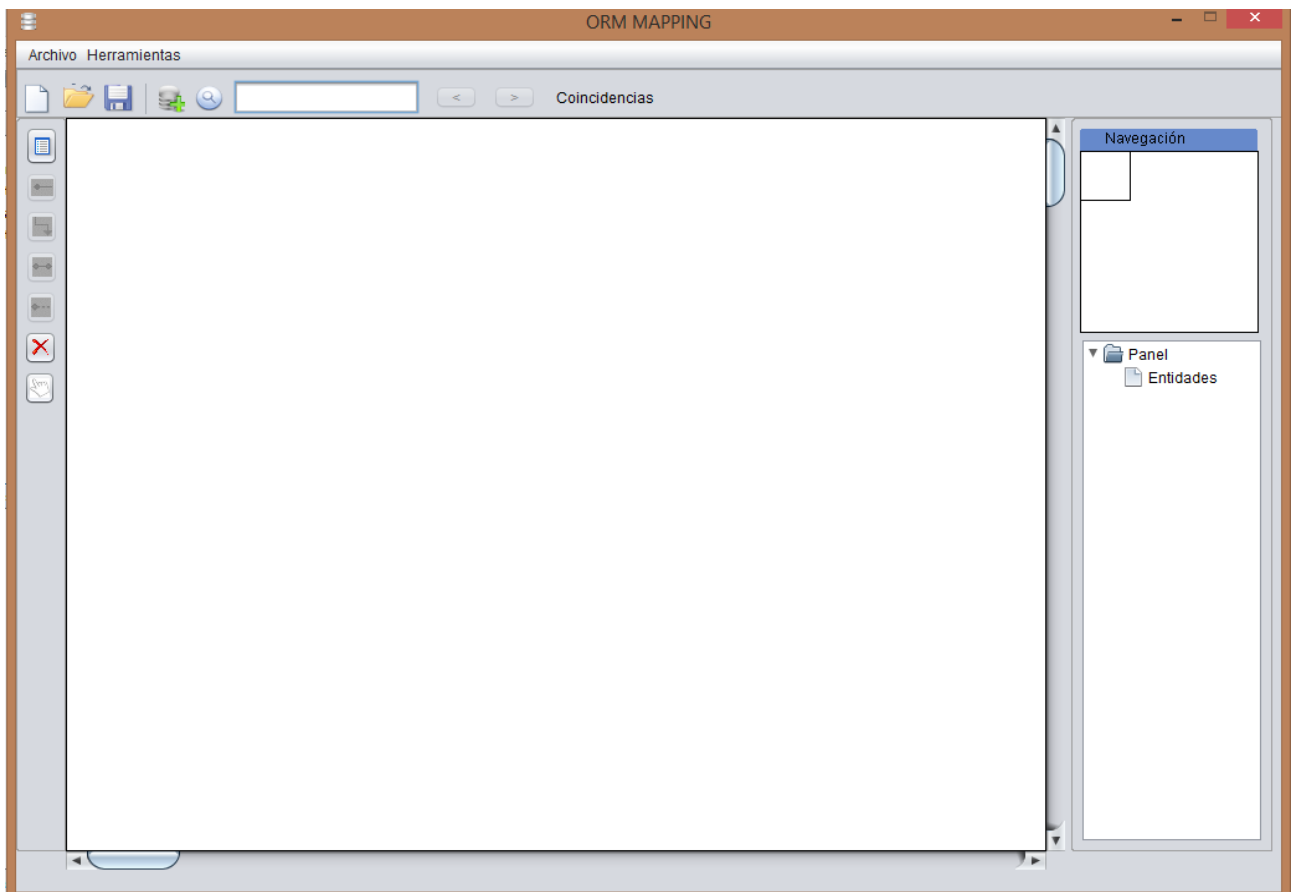


Figura 16. Interfaz principal de la aplicación ORM MAPPING. (Elaboración propia)

### 2.9 Conclusiones del capítulo

Como resultado del capítulo se generaron los artefactos de la fase de exploración, planificación, diseño e implementación de la metodología seleccionada. Se identificaron los requerimientos del sistema, obteniéndose así sus funcionalidades y un listado de requerimientos no funcionales a tener en cuenta para el desarrollo de la herramienta. Además, se aplicaron métricas para la validación de los requisitos demostrando que los mismos son en su totalidad estables y no ambiguos. La validación del diseño fue realizada a través de métricas arrojando resultados satisfactorios. Finalmente, se decidió implementar el sistema bajo una arquitectura de tres capas para satisfacer las características de la solución propuesta, minimizando la dependencia entre las mismas. Se utilizaron los estándares de codificación en el proceso de implementación, obteniéndose como resultado final la herramienta ORM MAPPING, que contribuye a la disminución del tiempo de realización de las tareas de los arquitectos de datos.

## Capítulo 3. Pruebas y validación de las variables de la investigación

### 3.1 Introducción

El presente capítulo está orientado a las pruebas y validación de las variables de la investigación que se llevan a cabo para dar total cumplimiento a los requisitos establecidos por el cliente. Se evidencian las pruebas aplicadas a la solución, con el objetivo de comprobar las funcionalidades en los diferentes escenarios y así verificar en todos los casos que los resultados sean los esperados. Terminada las mismas se exponen los resultados obtenidos mediante la prueba piloto ejecutada por los especialistas de CEGEL, demostrando el cumplimiento del problema planteado mediante el uso de ORM MAPPING.

### 3.2 Fase de pruebas

La verificación de los requisitos que debe cumplir la aplicación a través del análisis a las posibles combinaciones de entradas y de las salidas de datos, además de la comprobación de que el software trabaje como fue diseñado, son los objetivos que se persiguen con la realización de las pruebas al sistema. Como resultado permitirá un mayor control e identificación temprana de los defectos y fallos, garantizando la calidad del desarrollo con una reducción notable de los costos necesarios para corregir los errores (53). La calidad del sistema será medida en correspondencia con el número de no conformidades que sean detectadas.

En el presente subepígrafe se abordan los elementos analizados y los resultados obtenidos en la fase de aplicación de pruebas, para medir la variable independiente (proceso de mapeo de grandes volúmenes de clases del marco de trabajo Doctrine).

#### 3.2.1 Prueba de caja blanca

Para el desarrollo de estas pruebas se utilizó el framework JUnit, ya que se trata de un marco de trabajo de código abierto para la automatización de las pruebas de unidad de aplicaciones Java en los proyectos de software. Provee al usuario de herramientas, clases y métodos que facilitan la tarea de realizar pruebas en el sistema y así asegurar su consistencia y funcionalidad (54). Teniendo en cuenta lo antes mencionado, se decidió utilizarlo para realizar este tipo de pruebas por las características del sistema.

El proceso de pruebas de caja blanca se concentró principalmente en validar que cada segmento de código funcione apropiadamente. Además, cuenta con una interfaz simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada. Este tipo de prueba es totalmente objetiva y por lo tanto puede realizarla un ordenador de forma repetitiva, no depende



de la experiencia del programador (54). Para hacer uso de este marco de trabajo se definieron los siguientes pasos:

## Paso 1. Definición de los métodos a probar

Las pruebas se realizaron solo a las funcionalidades que determinan el correcto funcionamiento del sistema, entre ellas están:

Tabla 27. Métodos para las pruebas unitarias

Funcionalidades	
+exportarORM()	+ buscarTabla
+ exportarSQL()	+ validarNombreClase()
+ exportarImagen()	+ addUrlOrm()
+ sincronizarOrm()	+ addSchema()
+ sincronizarBd()	+ ajustarTabla()
+ crearTablas()	+ eliminarAtributo()
+ adicionarActualizarTable()	+ distanciaRelacion()
+ eliminarTablaRelacion()	
+ importarORM	

## Paso 2. Adaptación de los métodos al framework JUnit

Para explicar el proceso de realización de las pruebas de Caja Blanca con JUnit, fue preciso realizar algunas modificaciones que no cambian la lógica del negocio. Asimismo es necesario definir que los métodos testados sean de tipo *void*, identificar juegos de datos para probar cada método y agregar en la parte superior del método la anotación *@Test* (54).

Se tomó como objeto de estudio el método *adicionarActualizarTable ()*, función que se encuentra en la clase Sistema y que permite actualizar una tabla y adicionarle los atributos correspondientes, (ver figura 17).

```
public void adicionarActualizarTable() throws Exception {
    String nombre = "p";
    String nombreColumna = "a";
    boolean add = true;
    for (int j = 0; j < 3; j++) {
        if (nombre.equals("as")) {
            add = false;
            JOptionPane.showMessageDialog(null, "Ya existe un atributo con nombre p.", "Error", 0);
        } else if (nombreColumna.equals("m")) {
            add = false;
            JOptionPane.showMessageDialog(null, "Ya existe un atributo con el nombre de la columna a.", "Error", 0);
        }
    }

    if (add) {
    }
}
}
```

Figura 17. Código generado con JUnit en Netbeans. (Elaboración propia)

### Paso 3. 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 (54).

Se desarrollaron un total de 16 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 figura 18 aparecen los resultados de las pruebas.



Figura 18. Resultados de las pruebas con Junit. (Elaboración propia)

### 3.2.2 Prueba de caja negra

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca (40). Estas pruebas se aplicaron sobre los requisitos funcionales del sistema, con el objetivo de detectar errores a través de la respuesta del mismo, haciendo uso de los casos de prueba correspondientes a cada uno de estos.

#### Caso de prueba

Un caso de prueba incluye un conjunto de entradas, condiciones de ejecución y resultados esperados para conseguir un objetivo particular o condición de prueba por ejemplo verificar el cumplimiento de un requisito específico (55). Se diseñó una serie de casos de prueba para determinar así que los requisitos de software estaban completamente satisfactorios. A continuación se muestra un ejemplo de caso de prueba de uno de los requisitos del sistema Adicionar entidad, el resto de ellos se incluyen en el documento *0122\_Diseño de Casos de Pruebas Basado en Requisitos*.

Escenario	Descripción	Entidad	NameSpace	Tabla	Schema	Atributo	Tipo	Llave primaria	Null	Columna	Respuesta del sistema	Flujo central
EC 1.1 Adicionar entidad con datos correctos.	Adicionar los datos correctos a la entidad en el diagrama de clases persistentes.	V	V	V	V	V	V	V	V	V	El sistema crea la entidad con los datos adicionados.	El usuario selecciona la opción adicionar entidad en la paleta de herramientas.
		Persona	public	nuevo	public	id	integer	seleccionado	desmarcado	id		
EC 1.2 Adicionar entidad con datos incorrectos.	Adicionar datos incorrectos a la entidad en el diagrama de clases persistentes.	I	V	V	V	V	V	V	V	V	El sistema no permite insertar ese valor.	Selecciona la opción cerrar.
		12345ku	public	nuevo	public	nombre	text	seleccionado	desmarcado	nombre		
		V	I	V	V	V	V	V	V	V		
		Persona	1236**	nuevo	public	nombre	text	desmarcado	desmarcado	nombre		
		V	V	I	V	V	V	V	V	V		
Persona	public	*25lo	public	nombre	text	desmarcado	desmarcado	nombre				
EC 1.3 Adicionar entidad con datos incompletos.	Adicionar los datos incompletos a la entidad en el diagrama de clases persistentes.	I	V	V	V	V	V	V	V	V	El sistema muestra un mensaje de error porque el campo es obligatorio.	
		Campo vacío	public	nuevo	public	nombre	text	seleccionado	desmarcado	nombre		
		V	I	V	V	V	V	V	V	V		
		Persona	Campo vacío	nuevo	public	nombre	text	desmarcado	desmarcado	nombre		
		V	V	I	V	V	V	V	V	V		
		Persona	public	Campo vacío	public	nombre	text	desmarcado	desmarcado	nombre		
V	V	V	I	V	V	V	V	V				
Persona	public	nuevo	public	nombre	text	desmarcado	desmarcado	Campo vacío	El sistema crea la entidad con los datos adicionados.			

Figura 19. Caso de prueba Adicionar Entidad. (Elaboración propia)

## Descripción de las variables

Tabla 28. Descripción de las variables del requisito Adicionar entidad

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Entidad	Cadena	Obligatorio	Admite solo el formato [a-zA-Z_]{1}[a-zA-Z0-9_]*.
2	NameSpace	Cadena	Obligatorio	Admite solo cadena de texto.
3	Tabla	Cadena	Obligatorio	Admite solo el formato [a-zA-Z_]{1}[a-zA-Z0-9_]*.
4	Schema	Cadena	Obligatorio	Admite solo cadena de texto.

5	Atributo	Cadena	Obligatorio	Admite solo el formato [a-zA-Z_]{1} [a-zA-Z0-9_]*.
6	Tipo	Lista desplegable	Obligatorio	Admite los tipos de datos: Integer, character, character varying, text, doubleprecision, boolean, date
7	Llave primaria	Boolean	Opcional	Permite seleccionar si es o no llave primaria
8	Null	Boolean	Opcional	Permite seleccionar si es o no un valor nulo.
9	Columna	Cadena	Opcional	Admite solo el formato [a-zA-Z_]{1} [a-zA-Z0-9_]*.

## Resultados de las pruebas

El sistema luego de resueltas las 23 no conformidades detectadas, fue liberado por el Grupo Calidad CEGEL, lo cual se evidencia en el Acta de Liberación Interna de Productos Software (ver anexo 4). La siguiente tabla muestra el resultado obtenido durante las dos iteraciones de pruebas efectuadas. En ella se recoge la cantidad de requisitos que intervienen durante la ejecución de la prueba, así como el número de no conformidades, las cuales fueron resueltas al final de la iteración.

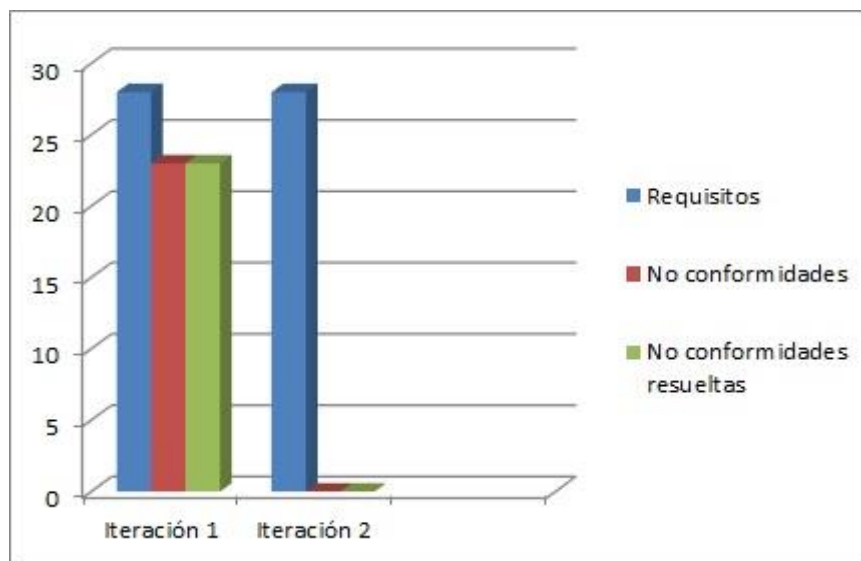


Figura 20. Resultado de las pruebas de Caja Negra. (Elaboración propia)

Las pruebas realizadas al sistema fueron satisfactorias desde el punto de vista interno y funcional, estas abarcaron requerimientos, funciones y la lógica interna del sistema. Se aplicaron los métodos de caja negra y caja blanca para verificar tanto la interfaz como el correcto funcionamiento interno de la herramienta. Combinar ambos enfoques permitió agilizar el proceso de pruebas al diseñar los casos de prueba.

## 3.2.3 Prueba para requerimientos de eficiencia

Según Pressman realizar pruebas a todos los caminos lógicos del software resulta impracticable. El propio autor propone seleccionar y comprobar un número limitado de rutas lógicas importantes (20). De acuerdo con lo antes descrito, se seleccionaron solo aquellos métodos que resultan de vital importancia para el funcionamiento del sistema.

De conjunto con los arquitectos de datos, usuarios finales de la aplicación se definieron dos hitos importantes en cuanto al rendimiento de la misma, en función del nivel de procesamiento que debe tener en estos dos momentos: el proceso de importar y exportar los ficheros de mapeo y la sincronización con la base de datos. Se estableció como cota máxima de tiempo de ejecución 20 y 3 segundos respectivamente para cada hito.

Para ejecutar las pruebas se seleccionó un computador con las siguientes características:

- **Hardware:** procesador core i3, frecuencia 3.3GHz, memoria ram 4GB.
- **Software:** gestor de base de datos PostgreSQL 9.1, Sistema Operativo Nova 4.0, máquina virtual de java 7.

Las pruebas se ejecutaron para ambos hitos con un número de entidades que oscilan entre 100 y 150 fuertemente relacionadas, atendiendo a las cantidades que modularmente se manejan en los proyectos de CEGEL.

Como resultado de las pruebas se obtuvieron los sigues datos.

Tabla 29. Resultados del tiempo de respuesta

Hito	Cota máxima	Tiempo medio obtenido
Importar y exportar los ficheros de mapeo	20 segundos	14.8 segundos
Sincronización con la base de datos	3 segundos	1.9 segundos

Atendiendo al comportamiento descrito en la tabla anterior, el sistema cumple con el requerimiento de tiempo de ejecución máximo definido a priori por los actores del sistema, garantizando que este se mantenga dentro de los umbrales necesarios para la explotación práctica de la herramienta.

Para darle cumplimiento al objetivo general de la investigación, además de las pruebas realizadas al sistema, se hizo necesario comprobar la variable dependiente de la situación problemática planteada y de esta forma conocer si se le dio solución al problema. En el epígrafe siguiente se explica cómo se llevó a cabo el procedimiento para dicha comprobación y los resultados obtenidos.

### 3.3 Validación de la investigación

Para la validación de la variable dependiente de la situación problemática (tiempo), fueron desarrolladas entrevistas (ver Anexo 3) y una prueba piloto a especialistas del proyecto SIGEF de CEGEL, que utilizan la herramienta Visual Paradigm para elaborar el diagrama de clases persistentes, generar las clases php con sus respectivos atributos y cargarlas en el IDE para su edición. Como resultado de las entrevistas se obtuvo que el tiempo de realización del proceso ORM, era variable en dependencia de la cantidad de clases que se manejan, características de la computadora en la que se trabaje y el nivel de experiencia del especialista.

Teniendo en cuenta estos indicadores, con una computadora que posea los requerimientos mínimos para realizar el proceso antes mencionado con esas herramientas tradicionales, un especialista con experiencia de no menos de tres años, demora alrededor de 40 horas en el diseño de más de 80 clases.

Sin embargo, al emplear la herramienta ORM MAPPING con el piloto realizado, se logró disminuir el tiempo de realización de dicho proceso a 20 horas. La misma se puso en práctica en el subsistema PDC, específicamente en el módulo Civil y Familia del proyecto antes mencionado, el cual cuenta con 93 entidades. Como consecuencia se reduce aproximadamente en un 50% el tiempo que le dedican los arquitectos de datos al mapeo de grandes volúmenes de clases con Doctrine.

### 3.4 Conclusiones del capítulo

En el capítulo quedaron presentados los elementos principales de las etapas de pruebas y validación del sistema. A partir de los resultados obtenidos, se demuestra que la solución propuesta cumple con los requisitos planteados por el cliente. Los resultados de las pruebas que fueron aplicadas verificaron una correcta ejecución del código y probaron que el sistema cumple con las funcionalidades requeridas. Finalmente, se comprobó el cumplimiento del problema a resolver, mediante la validación de las variables de la investigación con el uso de las entrevistas y la prueba piloto realizada por los especialistas del proyecto SIGEF de CEGEL; donde se obtuvo como resultado la disminución en un 50% del tiempo de realización del mapeo de grandes volúmenes de clases con Doctrine.

## Conclusiones generales

Con la realización y culminación del presente trabajo se arriba a las siguientes conclusiones:

- En la actualidad las herramientas de apoyo al proceso ORM poseen licencias de usuario final y el poco acceso de los arquitectos de datos de CEGEL a licencias válidas para el software privado de mapeo existente, dificulta el uso y soporte del mismo.
- Actualmente, el tiempo de realización del proceso ORM se ve afectado por la ejecución manual de algunas de sus actividades, por lo que la utilización de la herramienta ORM MAPPING permitirá disminuir los costos en el proceso de desarrollo.
- La implementación de la solución garantizó una aplicación funcional y sencilla de operar, que responde a los requisitos pactados con el cliente.
- Los resultados de las pruebas de software permitieron verificar la implementación de las funcionalidades desarrolladas, comprobando la calidad del sistema y demostrando que el mismo está listo para su uso.
- Las entrevistas y el piloto realizado por los especialistas de CEGEL, demostraron a partir de la utilización de la aplicación desarrollada, que se minimiza en un 50% el tiempo invertido por los arquitectos de datos en el proceso ORM mediante el marco de trabajo Doctrine.



## **Recomendaciones**

Se recomienda para futuras iteraciones y versiones del sistema propuesto:

- Ampliar el soporte para las conexiones a otros SGBD y otras herramientas ORM.
- Adicionar soporte para exportar archivos de mapeos en formato YAML y XML.
- Permitir importar diagramas de clases generados en Visual Paradigm en formato XML.

## Referencias bibliográficas

1. Marqués, M. *Bases de Datos*. Valencia, España : Publicacions de la Universitat Jaume I. Servei, 2011. ISBN/978-84-693-0146-3.
2. Sánchez, Jorge. *Diseño Conceptual de Base de Datos. Guía de aprendizaje*. California : s.n., 2004.
3. Osmel Yanes Enriquez, Hansel Gracia del Busto. *Mapeo Objeto / Relacional (ORM)*. La Habana : s.n., 2011. ISSN 1729-3804.
4. Christian Bauer, Gavin King. *Hibernate in Action. Practical Object/Relational Mapping*. s.l. : Manning Publications, 2004.
5. CEGEL, Centro de Gobierno Electrónico. Suite de Gestión de Proyecto. [Online] [Citado: 10 de Febrero de 2015.] <http://gespro.cegel.prod.uci.cu/>.
6. LIBROSWEB. *librosweb.es*. [Online] [Citado: 16 de Febrero de 2015.] [http://librosweb.es/libro/symfony\\_1\\_2/capitulo\\_8/por\\_qué\\_utilizar\\_un\\_orm\\_y\\_una\\_capa\\_de\\_abstraccion.html](http://librosweb.es/libro/symfony_1_2/capitulo_8/por_qué_utilizar_un_orm_y_una_capa_de_abstraccion.html).
7. *web.ontuts.com*. [Online] [Citado: 23 de Enero de 2015.] <http://web.ontuts.com/tutoriales/introduccion-a-object-relational-mapping-orm/>.
8. *Herramientas ORM para el desarrollo de aplicaciones en C++*. Milenis Fernández, Yassel Comas ,José Gabriel Espinosa. 12, La Habana : s.n., 2012, Vol. 5. 2306-2495.
9. Amador, Polo. *academia.edu*. *Estudio comparativo de sistemas de mapeo objeto relacional desarrollados en plataformas Open Source*. [Online] 5 de Enero de 2013. [Citado: 6 de Enero de 2015.] [http://www.academia.edu/9610912/Estudio\\_comparativo\\_de\\_sistemas\\_de\\_mapeo\\_objeto\\_relacional\\_desarrollados\\_en\\_plataformas\\_Open\\_Source](http://www.academia.edu/9610912/Estudio_comparativo_de_sistemas_de_mapeo_objeto_relacional_desarrollados_en_plataformas_Open_Source).
10. Raquel, Alvial Cid. *ORM - Object Relational Mapping (MapeoObjetoRelacional)*.
11. Alachisoft. *alachisoft.com*. [Online] [Citado: 16 de Febrero de 2015.] <http://www.alachisoft.com/resources/articles/orm.html>.
12. Carlos A Guerrero, Jose M Londoño. *ESTUDIO COMPARATIVO DE MARCOS DE TRABAJO PARA EL DESARROLLO SOFTWARE ORIENTADO A ASPECTOS*. [Online] 2014. [Citado: 1 de Febrero de 2015.] [http://www.scielo.cl/scielo.php?pid=S0718-07642014000200008&script=sci\\_arttext](http://www.scielo.cl/scielo.php?pid=S0718-07642014000200008&script=sci_arttext).

13. Pacheco, Nacho. *Doctrine 2 ORM Documentation* . 2011.
14. Guardado, Iván. web.Ontuts. [Online] 6 de Julio de 2010. [Citado: 26 de Enero de 2015.] <http://web.ontuts.com/tutoriales/utilizando-doctrine-como-orm-en-php/>.
15. Doctrine. doctrine2-preview-release. [Online] [Citado: 11 de 11 de 2014.] <http://www.doctrine-project.org/blog/doctrine2-preview-release.html>.
16. Visual Paradigm. [Online] [Citado: 6 de Enero de 2015.] <http://www.visual-paradigm.com/features/>.
17. *Free 64 bit programs*. [Online] [Citado: 6 de Enero de 2015.] <http://www.64bitprogramlar.com/portable-orm-designer-1-4-3-454-12268.html>.
18. Laukatu y PixelBinario. gentegeek.com. [gentegeek.com/sl-sp-ventajas-desventajas/](http://www.gentegeek.com/sl-sp-ventajas-desventajas/). [Online] [Citado: 11 de Enero de 2015.] <http://www.gentegeek.com/sl-sp-ventajas-desventajas/>.
19. Pressman, Roger S. *Software Engineering. A Practitioner's Approach. Seventh Edition*. New York : The McGraw-Hill Companies, 2010. ISBN 0-07-337597-7.
20. —. *Software Engineering: A Practitioner's Approach, 6/e*. 2005. ISBN 0072853182.
21. Sommerville, Ian. *Software Engineering (9th Edition)*. Boston : s.n., 2010. 978-0-13-703515-1.
22. Letelier, Patricio y Penadés, M. Carmen. *Métodologías ágiles para el desarrollo de software*. Valencia : Universidad Politécnica de Valencia, 2006.
23. Cruz, Dimelza del Pilar Remedios. *Guía de trabajo para las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP para el diseño de .* La habana : s.n., 2012.
24. Pressman, R. *Ingeniería de Software. Un enfoque práctico. Sexta Edición*. s.l. : Mc Graw Hil, 2005.
25. OSD. [Online] 2015. [Citado: 1 de Marzo de 2015.] <http://www.osdglobal.com/faq/desarrollo-software/comparativo-web-vs-escritorio>.
26. Larman, Craig. *UML Y PATRONES. Introducción al análisis y diseño orientado a objetos y al proceso unificado*. . España : s.n., 2004. 84-205-348-2.
27. Programación Desarrollo. *Qué es un entorno de desarrollo integrado, IDE*. [Online] [Citado: 13 de Eneo de 2015.] <http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide>.

- 
- 
28. NetBeans. [Online] [Citado: 20 de Enero de 2015.] <https://netbeans.org/community/releases/80/>.
29. Pérez, Teresa Garzón. *Sistema Gestores de Base de Datos*. PALMA DEL RÍO, CÓRDOBA : s.n., 2010. ISSN 1988-6047.
30. PostgreSQL. Comunidad Técnica de Desarrollo . [Online] 2015. [Citado: 18 de Febrero de 2015.] [http://postgresql.uci.cu/?page\\_id=30](http://postgresql.uci.cu/?page_id=30).
31. Mateu, Carles. *Desarrollo de aplicaciones web*. Barcelona : Fundación para la Universidad Oberta de Catalunya, 2004. ISBN: 84-9788-118-4.
32. Regina O. Obe, Leo S. Hsu. *PostgreSQL Up & Running. A PRACTICAL GUIDE TO THE ADVANCED OPEN SOURCE DATABASE*. Tokyo : O'Reilly, 2015. ISBN: 978-1-449-37319-1.
33. Fernandez, Oscar Belmonte. *Introducción al lenguaje de programación Java. Una guía básica*. España : s.n., 2004.
34. Ojeda, Francisco Charte. *Programacion GNU LINUX*. 2005.
35. Duarte, Manuel Palomo. *Programación en PHP a través de ejemplos*. Sevilla : s.n., 2013.
36. Eguiluz, Javier. *Desarrollo Agil Symfony 2. Primera edición*. 2011.
37. Date, C. J. *Introducción a los Sistemas de base de datos. Séptima Edición* . México : Addison Wesley Longman, Inc, 2001. ISBN: 968-444-419-2.
38. Lockhart, Thomas. *Tutorial de PostgreSQL*. s.l. : Postgres Global Development Group, 1999.
39. Universidad de Murcia. Universidad de Murcia. *Universidad de Murcia/~bmoros/Tutorial/parte13/cap13-1.html*. [Online] [Citado: 1 de Enero de 2015.] [dis.um.es/~bmoros/Tutorial/parte13/cap13-1.html](http://dis.um.es/~bmoros/Tutorial/parte13/cap13-1.html).
40. Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico 4ta Edición*. 2005.
41. Martínez, Juan Carlos Medina. *Análisis Comparativo de Técnicas, Metodologías y Herramientas de Ingeniería de Requerimientos*. D.F. México : s.n., 2004.
42. Karl.E.Wiegers. *Software Requirements* Microsoft Press. 1999.
43. Sánchez Fornaris. *Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas de Información Geográfica*. [Online] 2010. [Citado: 1 de Junio de 2014.]

[http://vinculando.org/articulos/sociedad\\_america\\_latina/propuesta\\_guia\\_de\\_medidas\\_para\\_evaluacion\\_sistemas\\_informacion.html](http://vinculando.org/articulos/sociedad_america_latina/propuesta_guia_de_medidas_para_evaluacion_sistemas_informacion.html).

44. Henríquez, Santiago Domingo Moquillaza. *Programación en N capas*. San Marcos : s.n., 2010. ISSN 1816-3823.
45. Escribano, Gerardo Fernandez. *Introducción a Extreme Programming*. 2002.
46. Vallespir, Diego. *CRC y un Taller*. Montevideo- Uruguay : s.n., 2002.
47. Gutierrez, Demián. *UML. Diagrama de clases*. Venezuela : s.n., 2011.
48. Erika Camacho, Fabio Cardeso,. *ARQUITECTURAS DE SOFTWARE. Guía de estudio*. 2004.
49. Escribano, Gerardo Fernández. *Introducción a Extreme Programming*. 2002.
50. Codigolinea.com. [Online] 2015. [Citado: 5 de Abril de 2015.] <http://codigolinea.com/2008/05/25/estilo-de-programacion-y-convencion-de-nombres-ii/>.
51. AMAP. *amap.cantabria.es*. [Online] 14 de Enero de 2014. [Citado: 20 de Abril de 2015.] <https://amap.cantabria.es/amap/bin/view/AMAP/CodificacionJava>.
52. Molina, Pedro J. *Prototipado rápido de interfaces de usuario*. España : s.n., 2002.
53. otros, Jorge Luis Valdez Mendoza y. slideshare.com. [Online] 2013. [Citado: 3 de Mayo de 2015.] <http://es.slideshare.net/abnergerardo/pruebas-de-sistemas-y-aceptacion-23663195>.
54. JUnit. [Online] 4 de Diciembre de 2014. [Citado: 9 de Mayo de 2015.] <http://junit.org>.
55. Aristegui, José Luis. *LOS CASOS DE PRUEBA EN LA PRUEBA DE SOFTWARE*. Chile : Revista Digital Lámpsakos,, 2010.