Universidad de las Ciencias Informáticas

Facultad 3

Título: Herramienta para la generación de consultas de Doctrine

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Roberto Alfonso Rivero

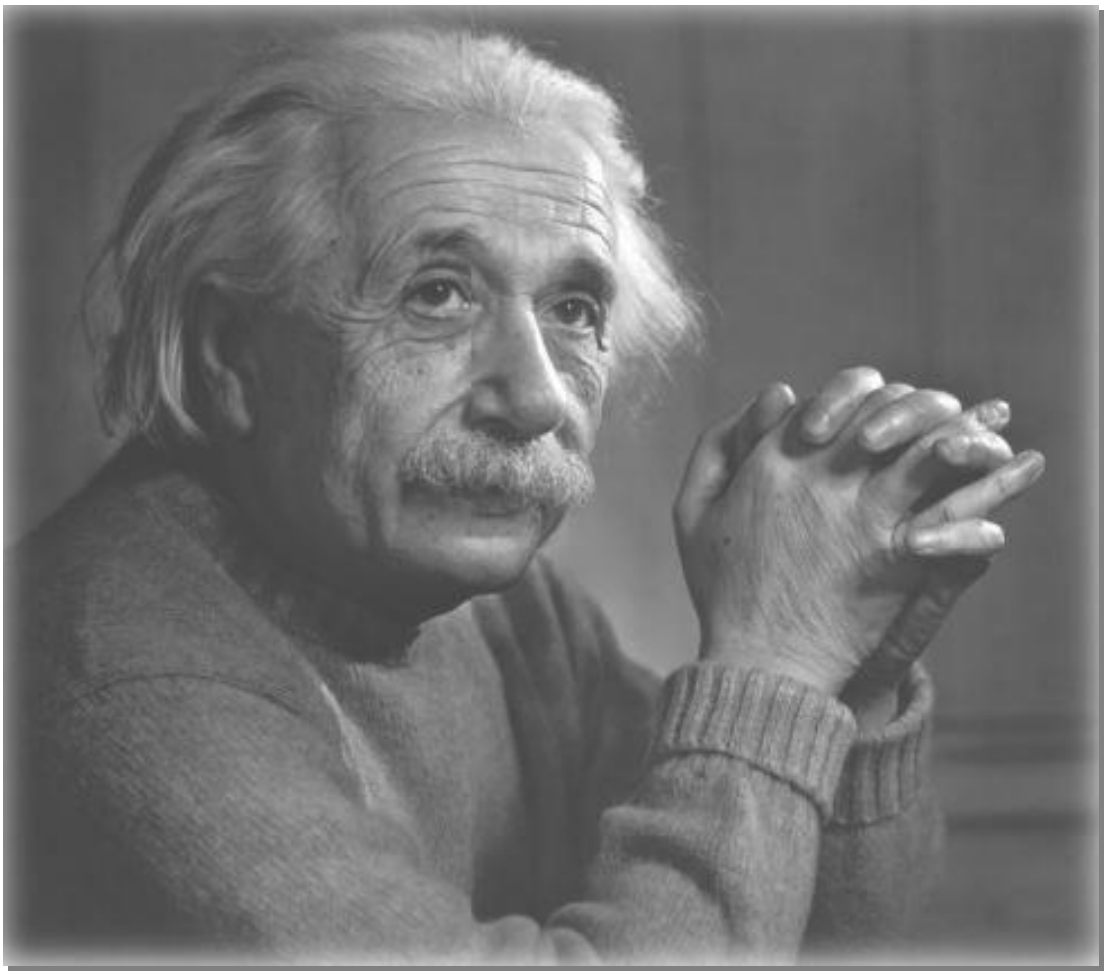
Gabriel Ibarra García

Tutores: Ing. Erich Mario Gómez Pérez

Ing. Inoelkis Velázquez Osorio

La Habana

Junio, 2015



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

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmamos la presente a los 19 días del mes de Junio del año 2015.

_____

Firma del Autor
Roberto Alfonso Rivero

_____

Firma del Autor
Gabriel Ibarra García

_____

Firma del Tutor
Ing. Erich Mario Gómez Pérez

_____

Firma del Tutor
Ing. Inoelkis Velázquez Osorio

DATOS DE CONTACTO

Nombre y apellidos del tutor(a): Ing. Erich Mario Gómez Pérez
Institución: Universidad de las Ciencias Informáticas.
Título: Ingeniero en Ciencias Informáticas.
Correo: **emgomez@uci.cu**

Nombre y apellidos del tutor(a): Ing. Inoelkis Velázquez Osorio
Institución: Universidad de las Ciencias Informáticas.
Título: Ingeniero en Ciencias Informáticas.
Correo: **inoelkis@uci.cu**

AGRADECIMIENTOS

De Roberto:

A mis padres por haberme apoyado y siempre haber estado a mi lado.

A mis hermanos por siempre estar ahí para mí cada vez que los necesitaba.

A mis sobrinos por siempre confiar en mí.

A mi familia completa por toda la atención brindada.

A mi novia por haberme ayudado tanto en los momentos difíciles.

A mis suegros y mi cuñado por guiarme siempre por un buen camino.

A mis compañeros de aula por el apoyo que siempre me brindaron.

A mi compañero de tesis por el empeño puesto para lograr llegar a este momento.

A mis amigos que han demostrado su verdadera amistad.

A mis tutores por toda la ayuda posible que nos brindaron, en especial Erich.

A todas las personas que de una forma u otra hicieron posible que llegara este día.

A todos los presente por haberme prestado su atención.

AGRADECIMIENTOS

De Gabriel:

Agradecer a mis padres Coralia del Pilar García y Félix Ibarra Tapia por haberse sacrificado tanto por mí, por estar siempre a mi lado apoyándome en las buenas y en las malas, dándome su amor y confianza en todo momento.

A mis abuelos, en especial mi abuela Mercedes, que siempre estuvo atenta en todo, a mis tías y tíos, por mantenerse siempre a mi lado dándome todo el apoyo que necesitaba para estar hoy donde estoy.

A mi hermano Javier y a mis primos, gracias por estar siempre a mi lado y darme el apoyo que siempre he necesitado.

A quienes nunca vacilaron para darme la mano en cualquier circunstancia.

A quienes con sus palabras y optimismo levantaban mi ánimo y me daban fuerzas para seguir adelante.

A los que me enseñaron todas las cosas que sé y me inculcaron valores importantes que me han ayudado a ser la persona que soy.

A mis tutores, por estar ahí en todo momento gracias, darme un poquito de su tiempo, y pasar malas noches ayudándonos. A Erick Mario, no solo fuiste tutor, sino fuistes un amigo, gracias por las muchas horas del día que nos dedicaste.

A todos muchas gracias.

DEDICATORIA

De Roberto:

A mis padres y mi hermano Albe por ser mi inspiración.

De Gabriel:

A mis padres que siempre me apoyaron y estimularon para terminar y vencer las adversidades que pudieron presentarse.

A mi hermano, y mi novia que siempre estuvieron a mi lado impulsándome a seguir adelante tanto en los momentos buenos, como malos.

En fin, a toda mi familia, por todo su amor y preocupación.

RESUMEN

La Universidad de las Ciencias Informáticas cuenta con diversos centros de producción, uno de ellos es el Centro para la Informatización de Entidades. En dicho centro, el Departamento de Desarrollo de Componentes, creó el marco de trabajo Sauxe, que es utilizado por varios sistemas informáticos como Cedrux, BK-Import/Export y SIPAC. Este presenta como deficiencia la complejidad en la forma de acceder a los datos mediante el Lenguaje de Consulta de Doctrine. Dicho lenguaje es muy complejo para los desarrolladores debido a que trae consigo un aumento de los errores en la creación de las consultas y a su vez un exceso del tiempo en el proceso de implementación de la capa de acceso a datos. Otra de las inconvenientes es la violación del estándar de codificación, trayendo consigo problemas a la hora del soporte y mantenimiento. Para solucionar dichos problemas, se crea la Herramienta para la generación de consultas de Doctrine, la cual permite la creación de consultas de selección, actualización, eliminación y funciones autogeneradas de forma visual, además de exportar en un fichero todas las funciones diseñadas por el desarrollador, disminuyendo el tiempo de implementación de la capa de acceso a datos de las aplicaciones desarrolladas empleando Doctrine.

Palabras claves: Acceso a Datos, Tiempo, Generación de consultas, Doctrine, Persistencia de los datos.

ÍNDICE

INTRODUCCIÓN

La sociedad actual se encuentra en un proceso de constante cambio y nuestro país enfrenta el reto de llevar a cabo la informatización de la sociedad cubana utilizando las nuevas Tecnologías de la Información y las Comunicaciones (TIC). Esta acción ha acelerado los procesos de cambio debido a que los equipos de trabajo crecen continuamente y los problemas son mayores, lo que conduce a desarrollar herramientas más complejas, como consecuencia de las exigencias de los clientes en cuanto a la calidad y tiempo de respuesta de las mismas.

Como consecuencia del crecimiento a nivel mundial de la informatización y a su vez el aumento en la generación y almacenamiento de la información, se generan grandes volúmenes de datos que no pueden ser procesados de forma manual, pues se haría el trabajo mucho más engorroso, y se afectaría la persistencia de la información. Para atender estas necesidades surge la tecnología de las bases de datos (BD), utilizada para manejar gran cantidad de información. En la actualidad el enfoque de BD es extensamente utilizado por ser una de las soluciones más factibles para manejar grandes volúmenes de datos.

En nuestro país, una de las instituciones que se dedica al desarrollo de soluciones informáticas es la Universidad de las Ciencias Informáticas (UCI), la cual desempeña un papel importante en el cumplimiento de este objetivo por la cantidad de tecnologías con la que cuenta y los logros alcanzados en sus años de existencia.

La UCI cuenta con diversos centros de producción, uno de ellos es el Centro para la Informatización de Entidades (CEIGE). En dicho centro, el Departamento de Desarrollo de Componentes, desarrolló el marco de trabajo Sauxe sobre el cual se desarrollan algunas de las soluciones del centro. Sauxe está sustentado por la integración de diversos marcos de trabajos como ExtJS, Zend Framework y Doctrine, contiene un conjunto de componentes reutilizables que logran una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. (Pupo, 2010)

Sauxe está compuesto por una arquitectura basada en niveles o capas. De estas capas una de las más importante es la Capa de Acceso a Datos, en esta se encuentra el Mapeador de Objeto Relacional (ORM)

Doctrine, el cual trabaja la persistencia para la comunicación con el servidor de datos, además, este separa la lógica del negocio al mapear los objetos de la aplicación a datos que persisten en una base de datos, creando un nivel de abstracción entre la aplicación y la base de datos. (Pupo, 2010)

Presenta como deficiencia la complejidad en la forma de acceder a los datos mediante el código DQL (Lenguaje de Consulta de Doctrine), debido a la compleja creación de consultas de forma manual, lo que aumenta el tiempo de culminación de las aplicaciones informáticas. El uso de este lenguaje requiere un amplio conocimiento del mismo debido a que sin el conocimiento suficiente, se pueden cometer errores en la creación de consultas provocando en ocasiones deficiencia a la hora de persistir los datos, así como al obtener el resultado esperado de la misma, trayendo todo esto demoras en la obtención de la solución deseada. La creación de funciones genéricas, ya sea a la hora de modificar, eliminar u obtener los datos se hace un proceso repetitivo, ya que es similar en todas las clases del dominio.
Otra de las inconvenientes es la violación del estándar de codificación, trayendo consigo problemas a hora del soporte y mantenimiento.

A partir de la problemática antes descrita se identificó como **problema a resolver:** ¿Cómo disminuir el tiempo de implementación de la capa de acceso a datos de las aplicaciones desarrolladas empleando Doctrine como marco de trabajo para la persistencia de los datos?

El problema antes expuesto está enmarcado en el proceso de generación de código fuente como **objeto de estudio.**

Con el propósito de darle solución al problema formulado, se establece como **objetivo general:** Desarrollar una herramienta que permita la generación de consultas de Doctrine para disminuir el tiempo de implementación de la capa de acceso a datos de las aplicaciones desarrolladas empleando Doctrine como marco de trabajo para la persistencia de los datos.

El objetivo general se desglosa en los siguientes **objetivos específicos:**

- ✓ Elaborar el Marco Teórico de la investigación relacionado con las herramientas de generación de código fuente.
- ✓ Realizar el análisis y diseño de la Herramienta para la generación consultas de Doctrine.

- ✓ Implementar la Herramienta para la generación consultas de Doctrine.
- ✓ Validar la solución propuesta mediante métricas para la validación del diseño, pruebas de caja blanca, caja negra y el pre-experimento.

Del objeto de estudio analizado, se puede definir como **campo de acción:** Herramientas de generación de código fuente.

La investigación parte de la siguiente **Idea a defender:** Si se desarrolla una herramienta que permita la generación de consultas de Doctrine se disminuirá el tiempo de implementación de la capa de acceso a datos de las aplicaciones desarrolladas empleando Doctrine, como marco de trabajo para la persistencia de los datos.

Métodos teóricos:

Los métodos teóricos permiten estudiar las características del objeto de investigación que no son observables directamente. De ellos, se utilizan en este trabajo los siguientes:

Histórico – Lógico

El método histórico estudia la trayectoria real de los fenómenos y acontecimientos en el de cursar. El método lógico investiga las leyes generales del funcionamiento y desarrollo de los fenómenos. Ambos métodos no están divorciados entre sí, sino que por el contrario, se complementan y están íntimamente vinculados. (Quintana Aput, 2014)

Este método científico, permitió realizar un estudio del estado del arte de las principales herramientas generadoras de consultas que existen en la actualidad.

Analítico – Sintético

El análisis es una operación intelectual que posibilita descomponer mentalmente un todo complejo en sus partes y cualidades. La síntesis es la operación inversa, que establece mentalmente la unión entre las partes, previamente analizadas y posibilita descubrir relaciones y características generales entre los elementos de la realidad. El análisis y la síntesis no existen independientemente uno del otro. (Quintana Aput, 2014)

La utilización de este método facilitó el estudio por separado de cada una de las herramientas generadoras de consultas utilizadas en la actualidad, determinando las características que presentan en común, con el fin de lograr una comparación entre ellas y valorar los resultados de interés para la investigación en curso.

Inductivo – Deductivo

La inducción es una forma de razonamiento por medio de la cual se pasa, del conocimiento de casos particulares, a un conocimiento más general, que refleja lo que hay de común en los fenómenos individuales. La deducción es una forma de razonamiento, mediante el cual se pasa de un conocimiento general, a otro de menor nivel de generalidad. (Quintana Aput, 2014)

Al aplicar este método se estudiaron los generadores de consultas, con el fin de definir sus características particulares, y basado en todas ellas se definieron, requisitos y cualidades que debe cumplir el sistema que se propone.

Métodos empíricos:

Los métodos empíricos describen y explican las características del objeto y representan un nivel de la investigación, cuyo contenido procede de la experiencia. De ellos se utilizan en este trabajo los siguientes:

Observación

Es el registro visual de lo que ocurre en una situación real, en un fenómeno determinado, clasificando y consignando los acontecimientos pertinentes de acuerdo con algún esquema previsto. (Quintana Aput, 2014)

Con su utilización se centró toda la atención en la situación actual existente en el Departamento de Desarrollo de Componentes y en los inconvenientes de la herramienta de generación de consultas DQL, a la hora de incluirla en el marco de trabajo del proyecto, por su implementación incompatible con los estándares de diseño y desarrollo de software seguidos por el proyecto.

La presente investigación tiene la siguiente estructura, **Introducción**, tres capítulos, **Conclusiones, Recomendaciones y Bibliografía**. A continuación se hace una descripción de que se va a tratar en cada uno de los capítulos:

Capítulo 1: “Marco Teórico”.

El contenido que emprende este capítulo es fundamentar el marco teórico de la investigación, así como el análisis de los principales conceptos y herramientas de generación de consultas utilizadas en la actualidad. Además se expondrán la metodología y el entorno de desarrollo a emplear para la creación del generador de consultas DQL.

En el **Capítulo 2:** “Análisis, Diseño e Implementación”.
El contenido que aborda este capítulo está relacionado con el análisis, diseño e implementación de la herramienta. El mismo incluye el modelo conceptual de la aplicación, los requisitos, tanto funcionales como no funcionales a tener en cuenta para su desarrollo, además del modelo de diseño y una breve descripción de las historias de usuario. Se validará el diseño mediantes métricas de validación del diseño y se tendrá en cuenta el estándar de codificación de Sauxe para la posterior implementación de la herramienta.

En el **Capítulo 3:** “Validación de la solución”.
Se validará la solución propuesta mediante los métodos de caja blanca, caja negra, comparación de consultas generadas y el pre-experimento para el correcto funcionamiento de la herramienta.

CAPÍTULO 1. MARCO TEÓRICO

Los generadores de consultas permiten crear consultas a través de una consola visual, posibilitando un mejor manejo con la base de datos y realizar varias consultas sin importar la complejidad de las mismas.
En este capítulo se describe el estudio del estado del arte que sustenta la investigación y varios conceptos asociados a este. Además se describen las tecnologías, herramientas y metodología a utilizar en la creación del generador de consultas DQL para el marco de trabajo Sauxe v2.2.

1.1 Marco conceptual

Para poder entender con mayor claridad lo expresado en el desarrollo del documento, se asumen los siguientes conceptos:

Persistencia de datos:

Es la capacidad que tiene un objeto de perdurar a través del tiempo de manera que este pueda ser accedido modificado o eliminado cuando el usuario lo desee. (Camps Riba, 2012)

Capa de Acceso a Datos:

En esta capa estará presente el ORM Doctrine, como marco de trabajo de persistencia para la comunicación con el servidor de datos mediante el protocolo Objeto de Datos de PHP (PDO), también estará un persistidor de configuración que es el encargado de comunicarse vía Lenguaje de Marcas Extensible (XML) con los ficheros de configuración del sistema. (Pupo, 2010)

Marco de trabajo:

Marco de trabajo (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" (Minnetto, 2007). En una estructura de soporte definida, en donde otro proyecto de software puede ser organizado y desarrollado, a partir de una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. 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 software. (Guerrero, Londoño, Suárez, & Gutiérrez, 2014)

1.2 Resumen de análisis bibliométrico documental.

Tabla 1: Resumen de bibliografías consultadas.

Tipo de Referencia Bibliográfica	Últimos 5 años	Años anteriores
Libros	6	7
Tesis de maestrías	5	0
Artículos en Revistas referenciadas en Web of Science, SCOPUS	1	1
Artículos publicados en la web	14	1
Conferencias	3	0

- Se consultaron 38 bibliografías, de las cuales 29 son de los últimos 5 años y las 9 restantes, de años anteriores, por lo que se llegó a la conclusión, que más del 40% de la bibliografía es de los últimos 5 años.
- Del total de la bibliografía investigada, más del 20% es en idioma inglés.
- De los libros que se escogieron después de haber hecho una investigación profunda sobre el tema, 6 fueron de los últimos 5 años y 7 de años anteriores, para un 47% actualizada y 53% restante menos actualizada.
- De las tesis de maestrías consultadas, que abordan sobre conceptos y contenido del trabajo de diploma, el 100% es de los últimos 5 años.
- Los artículos examinados en revistas como Scielo y IEEE[1] , ayudaron a comprender sobre métricas de validación y sobre marcos de trabajos existentes.

[1] **IEEE**: de las siglas en inglés **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers (Instituto de Ingeniería Eléctrica y Electrónica).

- Los artículos publicados en la web, facilitaron la comprensión de las herramientas utilizadas, para la creación de la herramienta generadora de consulta DQL. Las cuales el 93% está en los últimos 5 años y un 7% de las restantes.
- Las conferencias, facilitaron la comprensión del estándar de codificación y metodología utilizada para el desarrollo de la herramienta, la cual se encuentra en los últimos 5 años.

1.3 Herramientas generadoras de consultas para base de datos

1.3.1 Aqua Data Studio v15.0.0

El generador visual de consultas de Aqua Data Studio ofrece la posibilidad de construir consultas de bases de datos sin escribir instrucciones Lenguaje de Consulta Estructurado (SQL). Este permite seleccionar tablas de la base de datos directamente desde la interfaz gráfica de usuario, y luego especificar columnas que desea recuperar, uniones, opciones de ordenación, criterios de filtrado y otros parámetros de consulta en formato de cuadrícula. Junto con la selección de tablas y el agregado de parámetros de consulta, generará una instrucción SQL completa que puede verse, copiarse y ejecutarse desde la ventana del cuadro de diálogo. (System, 2014)

Funciones del generador visual de consultas (System, 2014)**:**

- **Agregar tablas y columnas a la consulta** – El generador visual de consultas permite incluir tablas y columnas en la consulta de manera automática. El panel SQL se actualizará instantáneamente para mostrar las columnas y los alias ingresados en forma de script para revisión. Las tablas usadas con más frecuencia pueden agregarse también a la barra de herramientas de accesos directos.
- **Crear JOIN (unión) con arrastrar y colocar** – Es posible crear JOIN para consultas arrastrando las columnas de la tabla de una entidad de tabla a otra en el panel del diagrama de tablas. El panel SQL muestra el tipo de JOIN y sus detalles de manera instantánea, a medida que se realizan los cambios.
- **Guardar archivos del generador de consultas** – Los archivos del generador de consultas (.xqb) pueden editarse en el editor de XML. Esta función es muy útil para compartir archivos .xqb entre usuarios que quizás no tengan los mismos nombres para sus conexiones de bases de datos pero se conecten al mismo servidor de base de datos y a la misma base de datos.
- **Guardar consultas** – Permite guardar consultas para editar posteriormente en el editor SQL de Aqua Data Studio, ya sea mediante la inclusión en un montaje en el explorador de scripts o como

parte de un directorio de scripts, asegurado a una conexión de servidor de base de datos dentro del explorador de esquema.

- **Ejemplos GROUP BY** – Es posible escribir rápidamente consultas GROUP BY seleccionando columnas y la función de agregado de la lista.

1.3.2 EMS SQL Manager para PostgreSQL v4.8

EMS SQL Manager for PostgreSQL es una poderosa herramienta gráfica para la administración y desarrollo de PostgreSQL Database Server (servidor de bases de datos PostgreSQL). Ofrece una gran variedad de herramientas poderosas para usuarios avanzados, tales como diseñador visual de base de datos, constructor visual de consultas, y un editor de objetos binarios. (EMS SQL MANAGER, 2015)

Características (EMS SQL MANAGER, 2015):

- Potente diseñador visual de base de datos.
- Excelentes herramientas visuales y de texto para elaboración de consultas.
- Administración efectiva de seguridad.

1.3.3 CODISA QUERY BUILDER v3.2

Es una herramienta para generar y compartir consultas y reportes a partir de las diferentes fuentes de datos con las que cuente su organización. (Codisa, 2014)

Principales Beneficios (Codisa, 2014):

- Permite extraer de forma rápida y sencilla la información requerida para la operación diaria de una compañía.
- Agiliza los procesos de comunicación de la información entre distintos miembros de una organización.
- Propicia mejoras en los procesos del negocio mediante el análisis fácil y oportuno de la información detallada por parte de las personas clave de una organización.
- Mantiene siempre actualizado el desempeño del negocio, accediendo a la información de una empresa de forma segura en cualquier momento y lugar.
- La empresa estará en capacidad de desarrollar y fomentar estándares corporativos para la presentación de la información.

Principales características (Codisa, 2014)**:**

- Poderoso e intuitivo generador de consultas y reportes.
- Administra la información disponible en las bases de datos.
- Envía reportes automáticos y periódicos a los miembros de la organización.
- Visualiza las consultas y reportes de múltiples maneras, de forma segura en cualquier momento y lugar.
- Administra la seguridad.

1.3.4 PgAdmin III

Es una aplicación de diseño y manejo de bases de datos para su uso con PostgreSQL. La aplicación se puede utilizar para manejar PostgreSQL 9.1 y funciona sobre casi todas las plataformas. Este software fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas SQL a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita la administración. La aplicación también incluye un editor de la sintaxis SQL, un editor de código del lado del servidor, un editor de consultas gráfico, el cual facilita al usuario generar el código SQL. (pgAdmin:PostgresSQL, 2014)

1.3.5 Resultado del estudio de las herramientas existentes

Durante el estudio de las herramientas, se obtuvieron similitudes entre ellas, que presentan un editor de consultas gráfico y todas son generadoras de código SQL. La mayoría de ellas son propietarias, no son basadas en tecnologías web y lo único que generan es código SQL, lo que constituye una desventaja para su empleo. Por lo que se llegó a la conclusión de que no satisfacen las necesidades de la investigación, ya que se requiere una herramienta para la construcción del código DQL, que reduzca el tiempo de creación de consultas, y facilite el desarrollo de aplicaciones en Sauxe. La Tabla 2 muestra la comparación entre las herramientas.

Tabla 2: Tabla comparativa de herramientas generadoras de consultas.

Indicador	pgAdmin	AquaFold	CodisaQB	SQL Manager
Tecnología	Desktop	Desktop	Desktop	Desktop

Código	SQL	SQL	SQL	SQL
Licencia	Libre	Propietaria	Propietaria	Propietaria
Sistema Operativo	Windows, Linux	Windows, Linux	Windows	Windows

1.4 Metodología de desarrollo

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo, en el que se definen productos de trabajos, roles y actividades, junto con prácticas y técnicas recomendadas.

Es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad, hasta que cumplimos el objetivo por el cual fue creado. (INTECO, 2010)

Para la realización de la aplicación se utilizó:

AUP-UCI: El Proceso Unificado Ágil de Scott Ambler (AUP), es una versión simplificada del Proceso Unificado Racional (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo (Rodríguez Sánchez, 2014):

- Desarrollo dirigido por pruebas (test driven development - TDD en inglés).
- Modelado ágil.
- Gestión de Cambios ágil.
- Refactorización de base de datos para mejorar la productividad.
- Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva.

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) se exige así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.

Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, para ello nos apoyaremos en el Modelo CMMI-DEV v1.3. Este modelo constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad. (Rodríguez Sánchez, 2014)

Fases AUP-UCI (Rodríguez Sánchez, 2014)**:**

- **Inicio:** Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente, que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto.
- **Ejecución:** En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto, considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
- **Cierre:** En esta fase se analizan, tanto los resultados del proyecto, como su ejecución y se realizan las actividades formales de cierre del proyecto.

Disciplinas AUP-UCI:

- Modelado de negocio (opcional).
- Requisitos.
- Análisis y diseño.
- Implementación.
- Pruebas interna.
- Pruebas de liberación.
- Pruebas de Aceptación.
- Despliegue (opcional).

Roles AUP-UCI (Rodríguez Sánchez, 2014)**:**

- Jefe de proyecto.
- Planificador.
- Analista.
- Arquitecto de información (Opcional)
- Desarrollador.
- Administrador de la configuración.
- Stakeholder (Cliente/Proveedor de requisitos).
- Administrador de calidad.
- Probador.
- Arquitecto de software (Sistema).
- Administrador de BD.

1.5 Tecnologías y herramientas para el desarrollo

Para el desarrollo de la herramienta generador de consultas DQL, se utilizaron varias herramientas y tecnologías definidas por el Departamento de Desarrollo de Componentes que facilitaron la creación del trabajo.

1.5.1 UML

Es un Lenguaje de Modelado Unificado (UML) basado en una notación gráfica, que permite: especificar, construir, visualizar y documentar los objetos de un sistema programado. Este modelado visual es independiente del lenguaje de implementación, de tal forma que los diseños realizados, usando UML, se puedan implementar en cualquier lenguaje que soporte sus posibilidades. Es uno de los lenguajes de modelado más utilizados para la definición de la arquitectura de una aplicación, que utiliza una gran variedad de diagramas (clases, secuencias, actividades, componentes, etc.) (UML, 2014)

1.5.2 PHP v5.3.3

PHP (Hypertext Preprocessor) es un lenguaje de código abierto interpretado, de alto nivel, embebido en páginas HTML. El lenguaje está orientado al desarrollo de aplicaciones web, que son interpretadas del lado del servidor. Sus sintaxis son muy similares a lenguajes como C y PERL. Puede ser utilizado en casi todos los sistemas operativos existentes, lo que permite migrar las aplicaciones de un sistema a otro sin necesidad de realizar cambios en el código. Su rapidez en la ejecución y los bajos requerimientos de consumo en los sistemas donde es desplegado, lo hace uno de los preferidos por los desarrolladores. Se integra

perfectamente a la mayoría de los Sistemas Gestores de Bases de Datos. Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso, además de poseer una comunidad de desarrolladores que intercambian experiencias, de esta forma, cuando se presenta un problema, es muy fácil obtener documentación para darle solución de manera rápida y sin costo. (PHP, 2014)

1.5.3 Doctrine v2

Doctrine es un potente y completo sistema ORM para PHP 5.2.3 +, que incorpora una Capa de Abstracción a Base de Datos (DBL). (Server Gove, 2010) Sauxe utiliza en la capa de acceso a datos el lenguaje DQL que implementa Doctrine. Entre otros elementos, se tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir, convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos. (Pacheco, 2011)

Ventajas que facilitan tareas comunes y de mantenimiento (Pacheco, 2011)**:**

Reutilización: La principal ventaja que aporta un ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.

Encapsulación: La capa ORM encapsula la lógica de los datos, pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.

Seguridad: Los ORM suelen implementar mecanismos de seguridad que protegen esta aplicación de los ataques más comunes como una Inyección SQL.

Mantenimiento del código: Gracias al correcto ordenamiento de la capa de datos, modificar y mantener este código es una tarea sencilla.

1.5.4 ExtJS. V2.2

ExtJS es una biblioteca JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas. Esta biblioteca incluye: Componentes Interfaz de Usuario (UI) de alto performance y personalizables. Modelo de componentes extensibles. Un API fácil de usar. (Sánchez Rosas, 2008) Sauxe utiliza ExtJS en la capa de presentación, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una

interfaz más amigable y funcional. Una de las grandes ventajas de utilizar ExtJS es que permite crear aplicaciones complejas, utilizando componentes predefinidos así como un manejador de diseños; gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en cada uno (Firefox, Internet Explorer, Safari, etc.). Además, la ventana flotante que provee ExtJS es excelente por la forma en la que funciona. Al moverla o redimensionarla solo se dibujan los bordes, haciendo que el movimiento sea fluido lo cual es una ventaja tremenda frente a otros. (Sencha inc., 2014)

1.5.5 Marco de trabajo Sauxe v2.2.

Sauxe es un marco de trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Además, emplea el patrón de arquitectura Modelo Vista Controlador (MVC) que separa los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos, también reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. (Pupo, 2010)

Su arquitectura está basada en cinco capas (Pupo, 2010):

1. Capa de Presentación: En esta capa se emplea las facilidades que brinda el marco de trabajo ExtJS para la construcción de interfaces amigables a la vista de los usuarios. ExtJS centra su desarrollo en tres componentes fundamentales JS-File, CSS-File y Client-page y Server-Page.

2. Capa de Control o Negocio: En esta capa se emplea el patrón de arquitectura MVC. De forma vertical al modelo descrito hasta este momento, estarán los aspectos fundamentales del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación Web.

3. Capa de Acceso a Dato: En esta capa estará presente el ORM Doctrine, como marco de trabajo de persistencia para la comunicación con el servidor de datos mediante el protocolo PDO, también estará un persistidor de configuración que es el encargado de comunicarse vía XML con los ficheros de configuración del sistema.

4. Capa de Dato: En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de ficheros de configuración de la arquitectura tecnológica.

5. Capa de Servicio: En esta última capa se encuentran todos los subsistemas que prestan y consumen servicios entre sí.

1.5.6 Servidor Web Apache v2.2

Es un servidor web HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Es una tecnología gratuita, de código abierto y altamente configurable de diseño modular por lo que resulta muy sencillo ampliar sus capacidades. Actualmente existen muchos módulos para Apache que son adaptables a este, y están disponibles para su instalación cuando sean necesarios. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor y es posible configurarlo para que ejecute un determinado script cuando esto suceda. (Foundation, 2012)

1.5.7 PostgreSQL 9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[2] y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. (PostgresSQL, 2013)

1.5.8 Visual Paradigm 8.0

Visual Paradigm para UML es una herramienta CASE[3] profesional que soporta el ciclo de vida completo del desarrollo de software. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Esta herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos UML. Permite modelado colaborativo con un Sistema de Control de Versiones (CVS) y Subversion, generación de código e ingeniería inversa, generación de bases de datos (transformación de diagramas entidad-relación en tablas de la base de datos), importación y exportación a ficheros XML, distribución automática de diagramas, entre otras características. (Visual Paradigm, 2014)

[2] BSD: Es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution).
[3] CASE: Herramienta CASE (***c****omputer* ***a****ided* ***s****oftware* ***e****ngineering*, ingeniería de software asistida por computadora).

1.5.9 NetBeans 8.0

El IDE NetBeans es un entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. Consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, empresariales, de escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript y Ajax, entre otros. (NetBeans, 2014)

Conclusiones parciales

En el capítulo se describió el estudio del estado del arte que sustenta la investigación y varios conceptos asociados a este. Se realizó una investigación de las herramientas generadoras de consultas que se utilizan en la actualidad, y se llegó a la conclusión de que se necesita desarrollar una herramienta que genere código DQL, ya que estas solamente generan código SQL. Además, fue definida la metodología a utilizar por ser ágil y flexible, siendo esta la utilizada por la UCI, estando certificada en el Modelo CMMI-DEV v1.3. Dado lo descrito anteriormente, se da cumplimiento al desarrollo del Marco Teórico de la investigación relacionada con las herramientas de generación de código fuente.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

En el presente capítulo se describe el diseño de la solución al problema planteado anteriormente, donde se verán los requisitos tanto funcionales como no funcionales a tener en cuenta para el desarrollo de la herramienta y una breve descripción de las historias de usuario. Además, se validará el diseño, mediantes métricas y se describirá el estándar de codificación que utiliza Sauxe para su implementación.
Todos los productos de trabajos generados en el presente capítulo se pueden consultar en el **Expediente de Proyecto,** que se encuentra en el Departamento de Desarrollo de Componentes.

2.1 Presentación de la solución

La Herramienta para la generación de consultas de Doctrine, cuyo objetivo es facilitar el trabajo y disminuir el tiempo de desarrollo de los programadores del Departamento de Desarrollo de Componentes, a la hora de crear consultas DQL para trabajar en el negocio de un sistema. Esta herramienta permitirá crear consultas seleccionando las cláusulas necesarias en dependencia del tipo de consultas que se necesite, las cuales pueden ser (Select, Update y Delete). Además, se podrán visualizar las consultas que se van generando para la aplicación que esté en desarrollo. Después de haber creado todas las consultas, esta brindará una opción de guardarlas en un fichero para su posterior uso.

2.2 Modelo conceptual

“Se llama modelo del dominio o conceptual a la representación visual de los conceptos u objetos del mundo real en un dominio de interés. Es el mecanismo fundamental para comprender el dominio del problema y para establecer conceptos comunes. ” (Larman, 2004)
Este muestra las clases conceptuales significativas en un dominio del problema, proporcionando un mejor entendimiento de los principales conceptos que se manejan en el negocio y ayuda a realizar este proceso de comprensión de la manera más cómoda (Larman, 2004). En la Figura 1 se representa el modelo de dominio establecido:

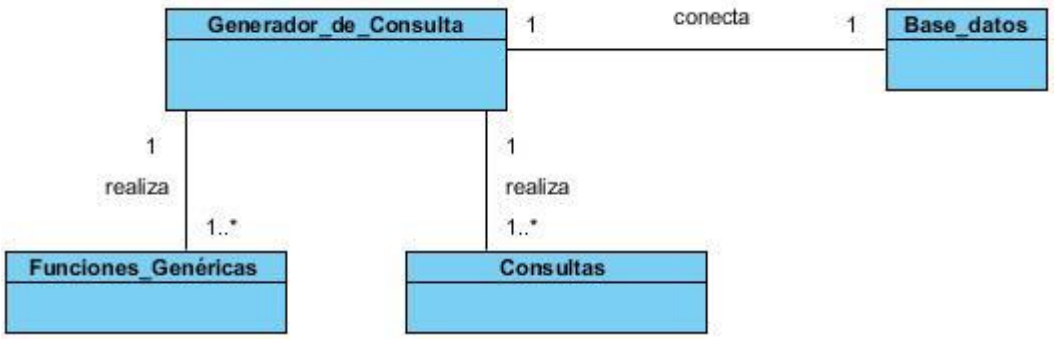


Figura 1: Diagrama del Modelo Conceptual.

Descripción de las entidades del modelo de dominio o conceptual:

Generador_de_Consulta: Se encarga de generar las funciones genéricas y consultas DQL.

Funciones_Genéricas: Las funciones genéricas se crean de acuerdo con las necesidades del cliente.

Consultas: Son creadas de acuerdo con la necesidad del cliente para su posterior generación.

Base_datos: Las base de datos con las cuales va a trabajar la herramienta.

2.3 Requisitos de la herramienta

2.3.1 Captura de requisitos

Para el proceso de captura de requisitos se utilizaron la técnica:

Tormenta de ideas: Esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos. (Ganesh, 2008) Para aplicar esta técnica se estructuró un equipo de trabajo conformado por:

- Jefe de Departamento de Desarrollo de Componentes.
- Analista principal del Departamento de Desarrollo de Componentes.
- Desarrolladores del Departamento de Desarrollo de Componentes.

Todos los integrantes del equipo de trabajo expusieran sus criterios e ideas sobre las principales funcionalidades, donde se obtuvieron los requisitos para la posterior implementación de la herramienta.

2.3.2 Requisitos funcionales

Los requisitos funcionales de un sistema describen la funcionalidad que se espera que éste proporcione. Son entendidos como capacidades que debe exhibir un sistema con el fin de resolver un problema. (Thayer & Merlin, 1997)

A continuación la Tabla 3 muestra los requisitos funcionales de la herramienta generadora de consultas DQL:

Tabla 3: Requisitos Funcionales.

No.	Nombre del Requisito Funcional
RF 1	Agrupación construir consulta.
RF 1.1	Listar entidad.
RF 1.2	Generar alias.
RF 2	Mostrar cláusulas de consulta DQL.
RF 3	Mostrar funciones genéricas.
RF 4	Listar visibilidad.
RF 5	Listar acción.
RF 6	Agrupación Gestionar parámetros.
RF 6.1	Adicionar parámetros.
RF 6.2	Eliminar parámetro.
RF 6.3	Eliminar todos los parámetros.
RF 6.4	Listar parámetros.
RF 7	Agrupación Gestionar atributos.
RF 7.1	Adicionar atributos.
RF 7.2	Eliminar atributo.
RF 7.3	Eliminar todos los atributos.
RF 7.4	Listar atributos.
RF 8	Agrupación Gestionar funciones de agregación.
RF 8.1	Adicionar función de agregación.
RF 8.2	Eliminar función de agregación.
RF 8.3	Eliminar todas las funciones.
RF 8.4	Listar funciones de agregación.
RF 9	Agrupación Gestionar modificación
RF 9.1	Adicionar campo a modificar.

RF 9.2	Eliminar campo a modificar.
RF 9.3	Eliminar todos los campos a modificar.
RF 9.4	Listar campos a modificar.
RF 10	Agrupación Gestionar unión entre entidades.
RF 10.1	Adicionar unión.
RF 10.2	Eliminar unión.
RF 10.3	Eliminar todas las uniones.
RF 10.4	Listar uniones.
RF 11	Agrupación Gestionar condición Where.
RF 11.1	Adicionar condición.
RF 11.2	Eliminar condición.
RF 11.3	Eliminar todas las condiciones.
RF 11.4	Listar condiciones.
RF 12	Agrupación Gestionar condición Having.
RF 12.1	Adicionar condición.
RF 12.2	Eliminar condición.
RF 12.3	Eliminar todas las condiciones.
RF 12.4	Listar condiciones.
RF 13	Agrupación Gestionar agrupación.
RF 13.1	Adicionar agrupación.
RF 13.2	Eliminar agrupación.
RF 13.3	Eliminar todas las agrupaciones.
RF 13.4	Listar agrupaciones.
RF 14	Agrupación Gestionar ordenar.
RF 14.1	Adicionar atributos a ordenar.
RF 14.2	Eliminar atributo a ordenar.
RF 14.3	Eliminar todos los atributos a ordenar.
RF 14.4	Listar atributos a ordenar.

RF 15	Visualizar consulta DQL.
RF 16	Guardar consulta DQL.

2.3.3 Requisitos no funcionales

Los requisitos no funcionales complementan a los funcionales y describen las condiciones ambientales y las cualidades necesarias para que el producto sea eficaz (PMBOK, 2013), además son definidos como requerimientos que deben dar respuesta a la operatividad del sistema final. (Stellman & Greene, 2005)
Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, como por ejemplo: requisitos de apariencia, de usabilidad, de rendimiento, de mantenibilidad y portabilidad entre otros.

Requisitos del Software

RNF1: La herramienta requiere las siguientes aplicaciones instaladas para su ejecución de parte del cliente:

- Navegador Mozilla Firefox en la versión 36 o superior.
- Sistemas Operativos GNU/Linux.

RNF2: La herramienta requiere las siguientes aplicaciones instaladas para su ejecución de parte del servidor:

- Sistemas Operativos GNU/Linux.
- Apache 2.2 o superior con módulo PHP 5.3.3 disponible y la extensión “pgsql” incluida.
- Un servidor de base de datos PostgreSQL 9.1

Requisitos del Hardware

RNF3: Requisitos mínimos que se necesitan para el correcto funcionamiento de la herramienta:

- Memoria RAM de 1GB o superior.
- Microprocesador a 1.64 GHz.

Requisitos de rendimiento

RNF4: El tiempo de respuesta de la creación del fichero en formato php no será mayor de 5 segundos.

Requisitos de usabilidad

RNF5: La herramienta es de fácil uso para los programadores que tengan un básico o avanzado conocimiento en creación de consultas DQL y Doctrine.

2.4 Historia de usuario

Las Historias de Usuario (HU), son descripciones cortas de una necesidad de un cliente de software. Su utilización es común cuando se aplican en marcos de trabajo ágiles. (PMOinformatica, 2014)

Las historias de usuarios son técnicas utilizadas para describir los requisitos del software, en ellas se describe de forma sencilla y clara, las actividades y acciones de los requisitos del sistema. Otra de sus características es que solamente proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo con llevará su implementación. Su nivel de detalle debe ser el mínimo posible, de manera que permita hacerse una ligera idea de cuánto costará implementar el sistema. A continuación las Tablas 4 y 5 describen las principales historias de usuarios de la herramienta generadora de consultas de Doctrine:

Tabla 4: Historia de Usuario "Visualizar el código DQL"

Historia de Usuario	
Número: HU17	**Nombre del requisito:** Visualizar el código DQL.
Programadores: Gabriel Ibarra García y Roberto Alfonso Rivero.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.	**Tiempo Real:** Cuatro semana.
Descripción: La historia de usuario permite visualizar el código DQL que se esté creando, utilizando cada una de las cláusulas y funciones necesarias.	
Observaciones:	
Prototipo de interfaz: Figura 2.	

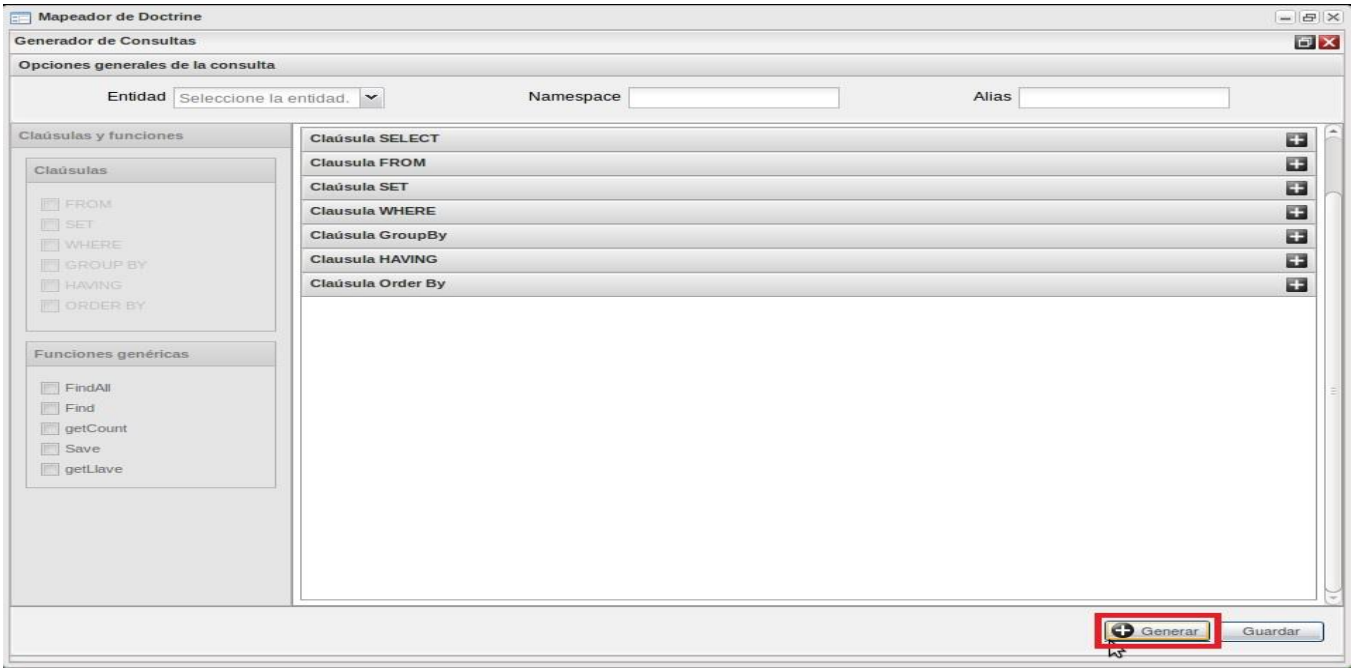


Figura 2: Prototipo de interfaz de la historia de usuario "Visualizar el código".

Tabla 5: Historia de Usuario "Guardar consulta DQL"

Historia de Usuario	
Número: HU18	**Nombre del requisito:** Guardar consulta DQL.
Programadores: Gabriel Ibarra García y Roberto Alfonso Rivero.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Catorce días.
Riesgo en Desarrollo: Alta.	**Tiempo Real:** Dos semana.
Descripción: La historia de usuario permite guardar un fichero en php, las consultas DQL creadas, las cuales van a estar contenidas las funciones con sus consultas DQL de la clase que se especificó en el campo entidad, namespace y alias.	
Observaciones:	
Prototipo de interfaz: Figura 3.	

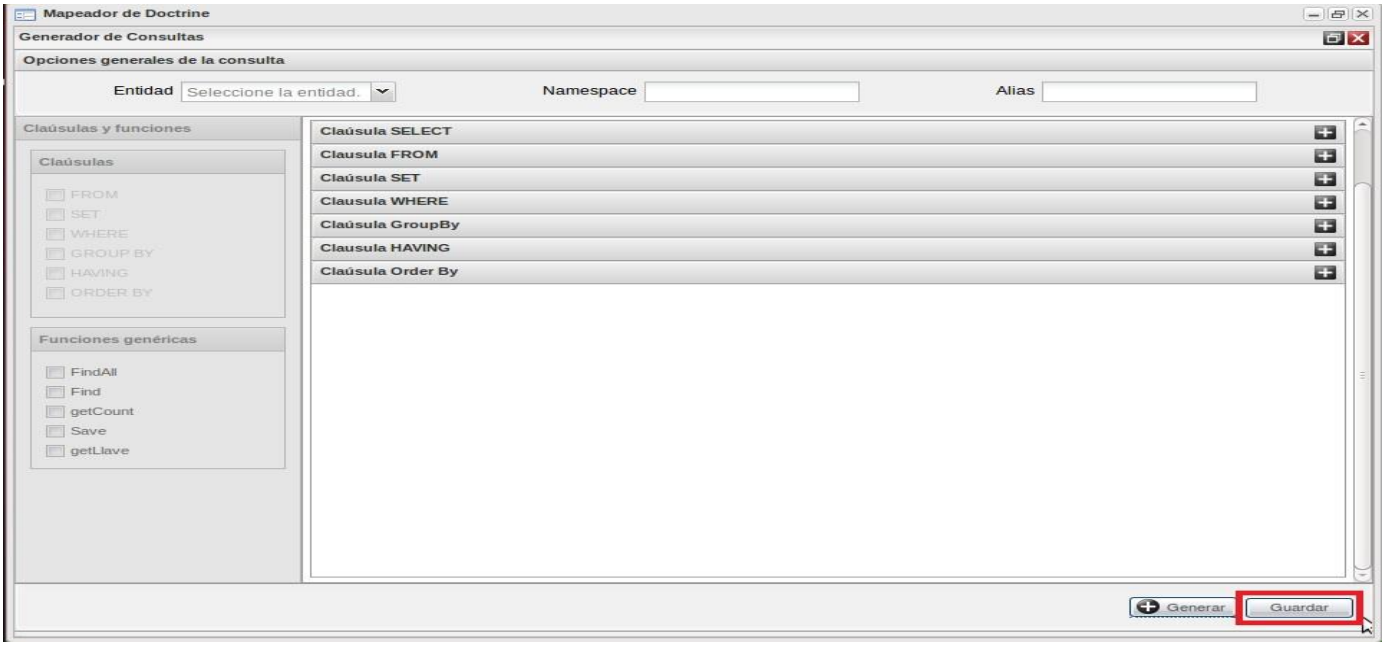


Figura 3: Prototipo de interfaz de la historia de usuario "Guardar consulta DQL".

2.5 Validación de los requisitos

La validación de los requisitos se realiza con la finalidad de comprobar que los requerimientos identificados sean precisos, consistentes, realistas, verificables, definan lo que el usuario desea del producto final, que los errores que hayan sido detectados sean corregidos y el resultado del trabajo cumpla con los estándares establecidos para el proceso, el proyecto y el producto. (Pressman R. , 2010)

Para la validación de los requisitos fueron aplicadas las siguientes técnicas (Sommerville, 2007):

Revisión técnica formal: La revisión de los requerimientos de la herramienta Generadora de Consultas fue realizada con los proveedores de requisitos. Con la utilización de esta técnica se validó que no existieran errores en el contenido o malas interpretaciones, información incompleta, inconsistencias y que los requisitos no fueran contradictorios, imposibles o inalcanzables, dando como resultado que fueran aprobados los que estaban descritos de forma correcta, clara y consistente.

Construcción de Prototipos: A partir de las historias de usuarios fueron conformados prototipos de interfaz de usuario. Con esto se validó que los requerimientos estaban en concordancia con las necesidades plasmadas por los proveedores de requisitos. El empleo de esta técnica ofreció como resultados que el

cliente tuviera una idea de la estructura de la interfaz de usuario y se favoreciera la comunicación con el mismo, ya que tenía una visión inicial de la herramienta.

Generación de casos de prueba: Fueron definidos y diseñados casos de pruebas para cada requisito especificado, con el objetivo de verificar el cumplimiento de los mismos. La utilización de esta técnica ofreció los siguientes resultados: fueron identificados los posibles escenarios de los requisitos, así como los datos de los campos determinados en estos, se validaron y aprobaron los que estaban bien enunciados, descritos y consistentes.

2.6 Diseño

2.6.1 Mecanismos de diseño

En la Herramienta para la generación de consultas de Doctrine se definieron los siguientes mecanismos de diseño, los cuales se muestran en las Figuras 4, 5 y 6:

Mecanismo de diseño para las páginas cliente

Figura 4: Mecanismo de diseño para las páginas cliente.

ext-all.js: Se encuentra entre las clases que posee el ExtJS y es la encargada de crear los componentes visuales de la vista.

ucid-all.js: Encargada de mostrar la interfaz estándar.

ext-base.js: Es la encargada de manejar las solicitudes, respuestas y componentes de ExtJS.

FileUploadField.js: Su función es la de cargar los ficheros.

Mecanismo de diseño para las clases controladoras

Figura 5: Mecanismo de diseño para las clases controladoras.

Todas las clases controladoras, definidas en el diseño propuesto, heredan de la clase ZendExt_Controller_Secure, ya que en ella se incluyen numerosas funcionalidades comunes en todas las controladoras.

Mecanismo de diseño para las clases modelos.

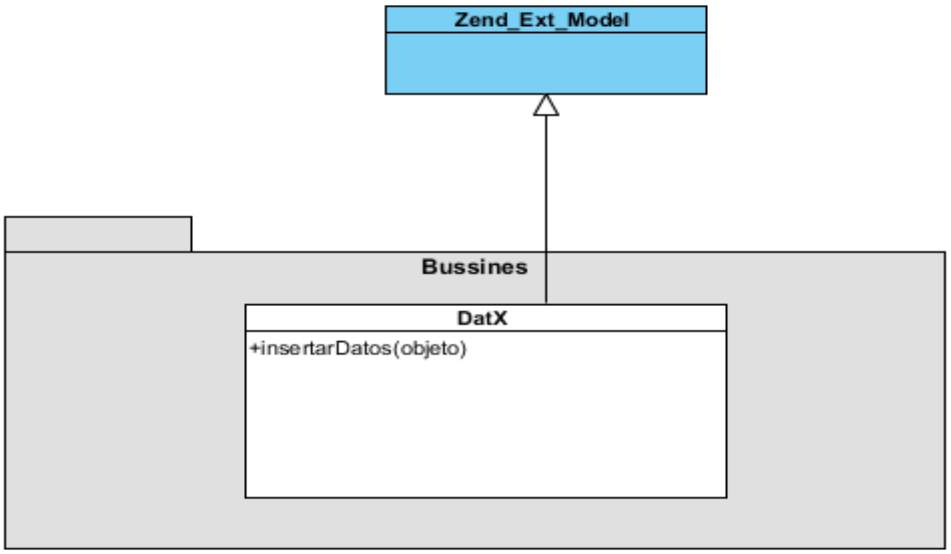


Figura 6: Mecanismo de diseño para las clases modelos.

Todas las clases modelos o model, definidas en el diseño, heredan de la clase ZendExt_Model, ya que ésta incluye las principales funciones para el manejo de los datos.

2.6.2 Diagrama de clases

El diagrama de clases recoge las clases de objetos y sus asociaciones. El objetivo principal de este modelo es la representación de los aspectos estáticos del sistema, utilizando diversos mecanismos de abstracción. (Cillero, 2014)

El diagrama de clases del diseño se realiza en la disciplina análisis y diseño para utilizarlo como referencia en la disciplina implementación. A continuación las Figuras 7, 8 y 9 muestran los diagramas de las consultas Delete, Select y Update respectivamente.

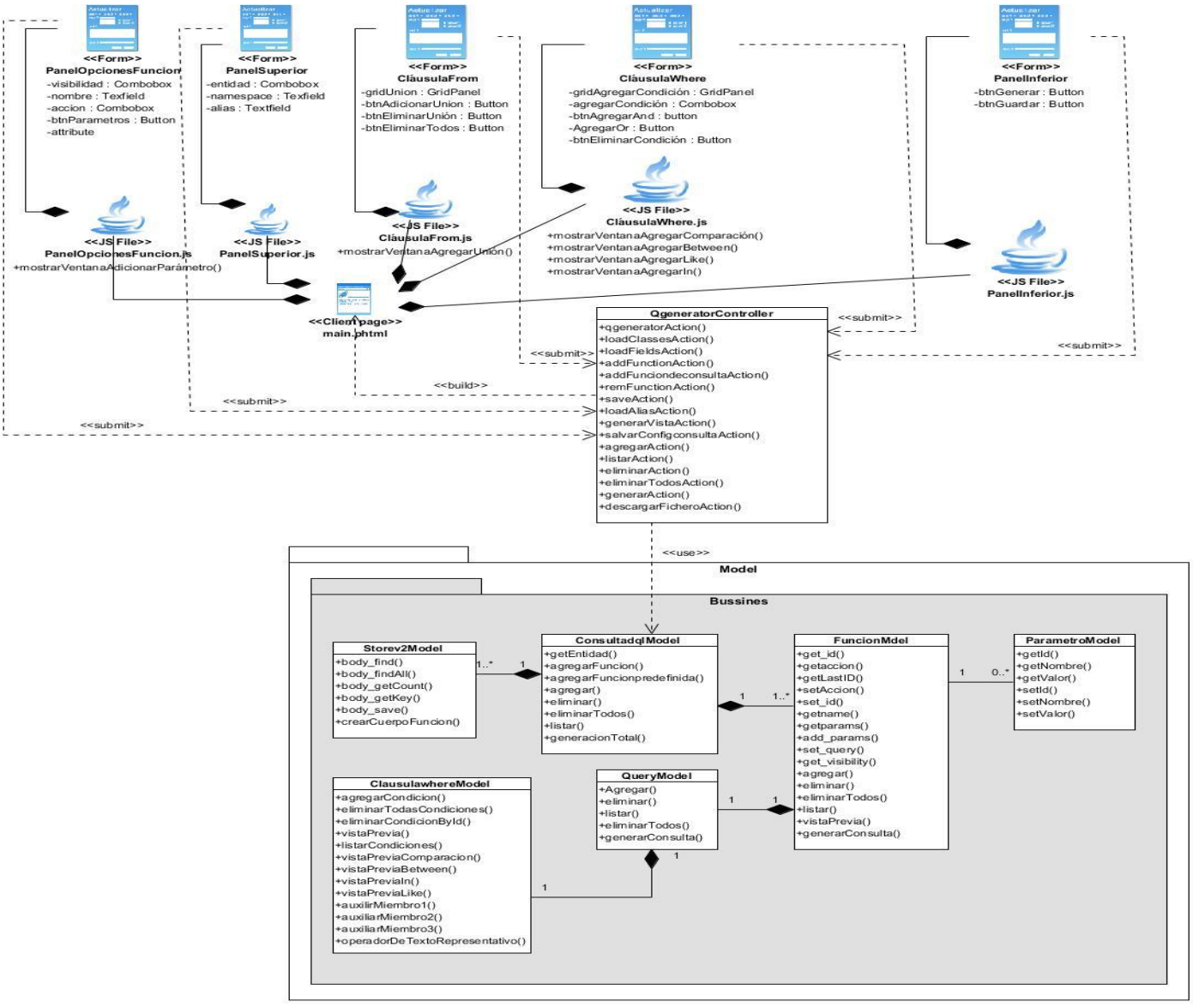


Figura 7: Diagrama de clases consulta Delete.

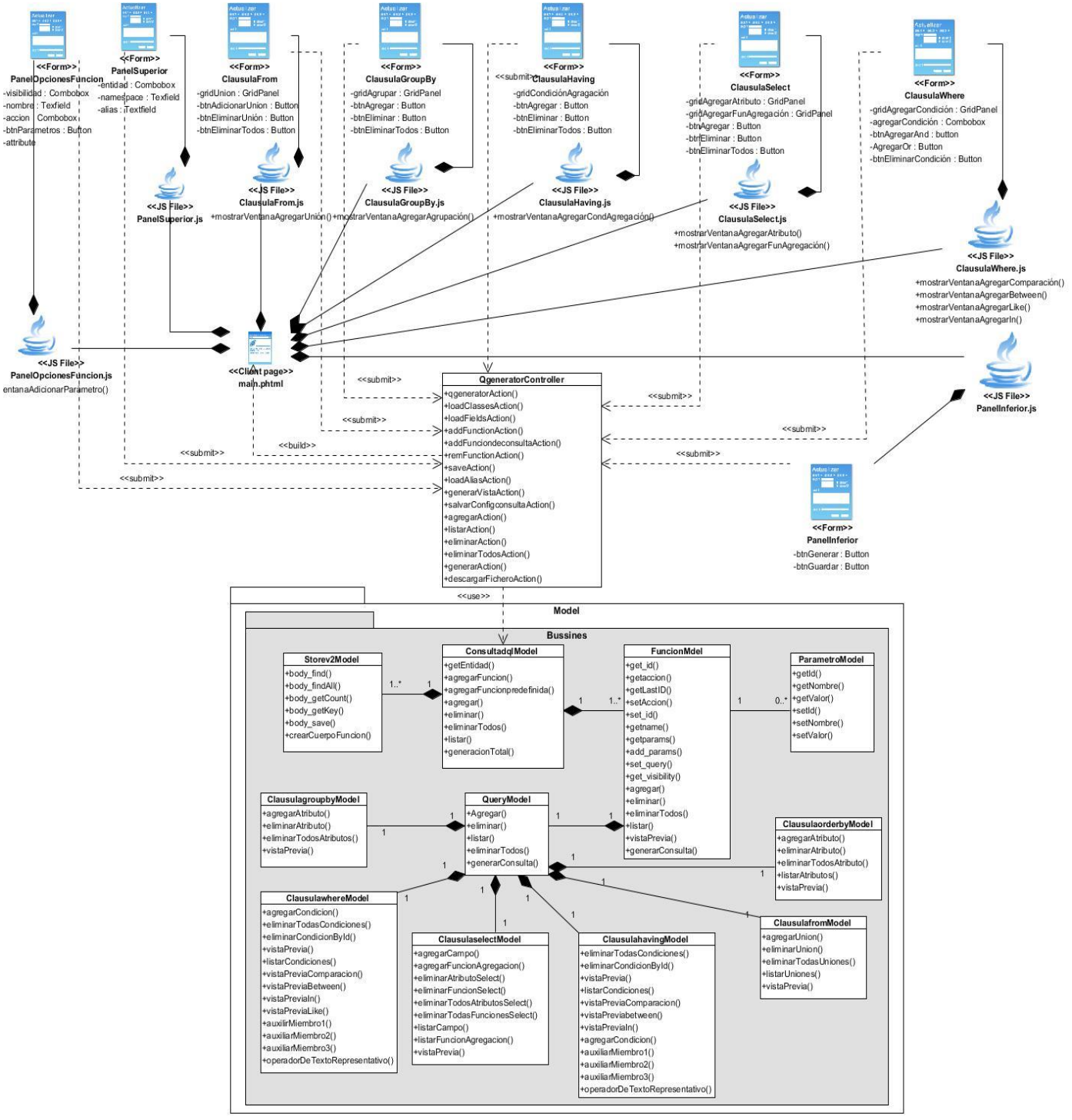


Figura 8: Diagrama de clases consulta Select.

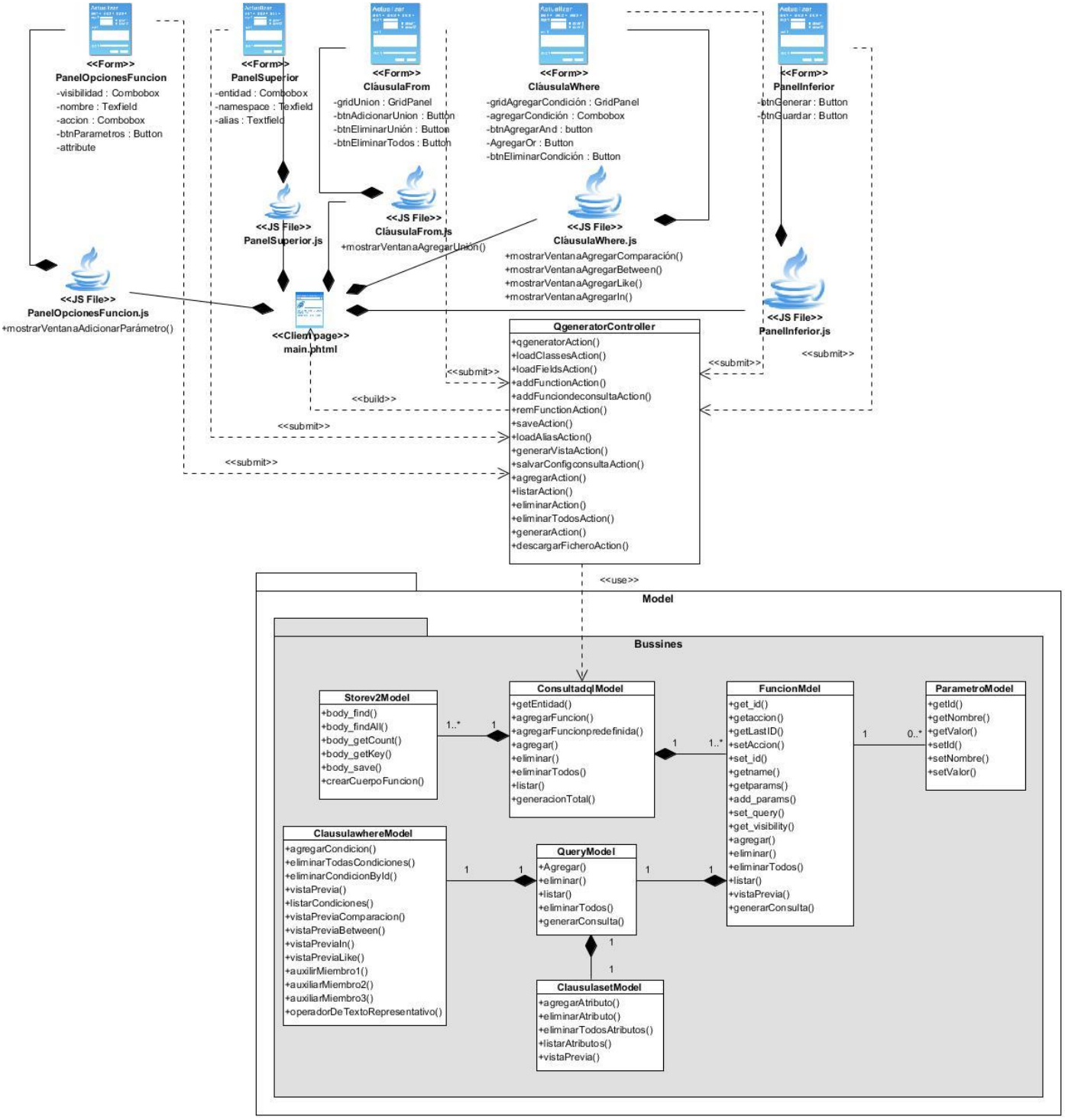


Figura 9: Diagrama de clases consulta Update.

2.6.3 Patrones arquitectónicos

2.6.3.1 Modelo-Vista-Controlador

El patrón MVC es un patrón de arquitectura de software encargado de separar la lógica de negocio de la interfaz del usuario y es el más utilizado en aplicaciones web, ya que facilita la funcionalidad, mantenibilidad y escalabilidad del sistema, de forma simple y sencilla, a la vez que permite “no mezclar lenguajes de programación en el mismo código”. (Bahit, 2011)

MVC divide las aplicaciones en tres niveles de abstracción:

- **Modelo:** representa la lógica de negocios. Es el encargado de acceder de forma directa a los datos actuando como “intermediario” con la base de datos.
- **Vista:** es la encargada de mostrar la información al usuario de forma gráfica y “humanamente legible”.
- **Controlador:** es el intermediario entre la vista y el modelo. Es quien controla las interacciones del usuario solicitando los datos al modelo y entregándolos a la vista para que ésta, lo presente al usuario, de forma “humanamente legible”.

Este patrón arquitectónico se utiliza en la creación de la Herramienta para la generación de consultas de Doctrine, debido a que su utilización fue definida por el CEIGE, además de estar implementado en el marco de trabajo Sauxe, sobre el cual se desarrollada la herramienta informática.

2.6.4 Patrones GRASP

Los Patrones Generales de Software para Asignación de Responsabilidades (GRASP) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. (Larman, 2004)

2.6.4.1 Experto

Este patrón fue usado en todas las clases definidas, ya que cuentan con la información necesaria para cumplir con la responsabilidad asignada.

2.6.4.2 Controlador

El empleo de este se puede observar en la creación de una clase controladora, para el proceso que se realiza en la herramienta generadora de consultas. Para controlar la generación de las consultas, fue creada la clase **QgeneratorController** que se muestra en la Figura 10.

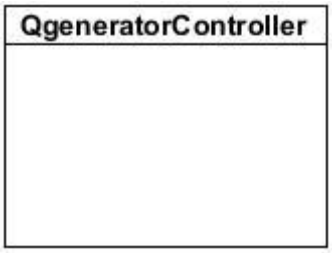


Figura 10: Patrón controlador.

2.6.4.3 Alta Cohesión

Se asignaron responsabilidades a todas las clases de forma tal que la cohesión siguiera siendo alta, ya que cada clase se encargará de realizar solamente las funciones que estén en correspondencia con la responsabilidad que posea. Este patrón se evidencia en todas la clases definidas, la Figura 11 muestra un ejemplo.

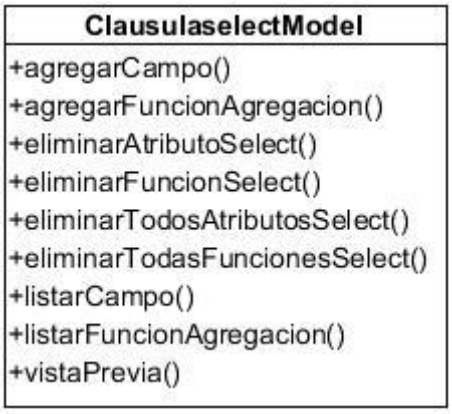


Figura 11: Patrón alta cohesión.

2.6.4.4 Bajo Acoplamiento

Este patrón expresa que entre las clases deberán existir pocas ataduras, es decir, estarán lo menos relacionadas posible, de forma tal que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, incrementando la reutilización, y disminuyendo la dependencia entre ellas. Este patrón se evidencia en todas la clases definidas, la Figura 12 muestra un ejemplo.

2.6.4.5 Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. La Figura 12 evidencia el patrón, donde la clase QueryModel registra las instancias de las demás clases representadas.

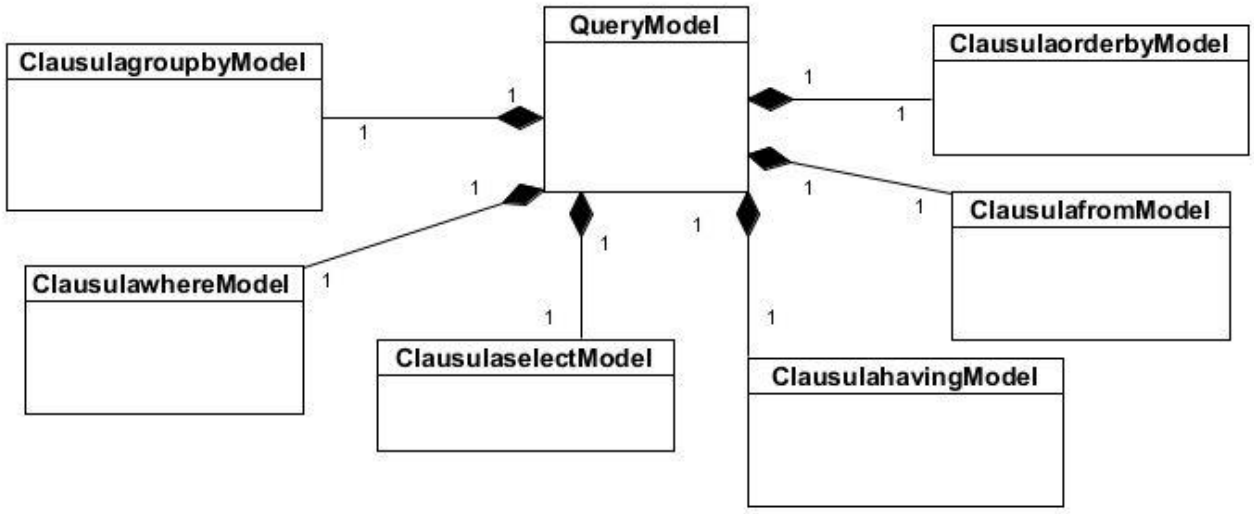


Figura 12: Patrón creador y bajo acoplamiento.

2.6.5 Métricas para evaluar el diseño propuesto

IEEE define como métrica "*una medida cuantitativa del grado de que un sistema, componente o proceso posee un atributo dado".* (IEEE, 2007)

Pressman plantea que la calidad del diseño se puede evaluar aplicando métricas básicas para el diseño orientado a objetos. (Pressman R. , 2005) Los atributos de calidad a evaluar son (Baryolo, 2010):

- Responsabilidad: es la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- Complejidad de implementación: grado de complejidad que posee la implementación de un diseño de clases específico.

- Reutilización: grado de reutilización presente en una clase o estructura de clases, dentro de un diseño de software.
- Acoplamiento: las conexiones físicas entre los elementos del diseño orientado a objeto, representan el acoplamiento dentro de un sistema orientado a objeto.
- Complejidad del mantenimiento: grado de esfuerzo necesario a realizar, para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- Cantidad de pruebas: número o grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

Se evaluó el diseño mediante las métricas Tamaño Operacional de Clases y Relaciones entre Clases que fueron propuestas por Lorenz y Kidd, estas son descritas a continuación (Pressman R. , 2010):

Tamaño Operacional de Clases (TOC): está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- **Responsabilidad:** Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación:** Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** Un aumento del TOC implica una disminución del grado de reutilización de la clase.

La Tabla 6 muestra el cálculo de los atributos de calidad.

Tabla 6: Cálculo de los atributos de calidad de la métrica TOC.

	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

La Figura 13 muestra los resultados obtenidos agrupados en los intervalos definidos. La gráfica refleja el comportamiento de la cantidad de clases contra procedimientos, lo que demuestra que el funcionamiento general del componente está distribuido entre las diferentes clases.

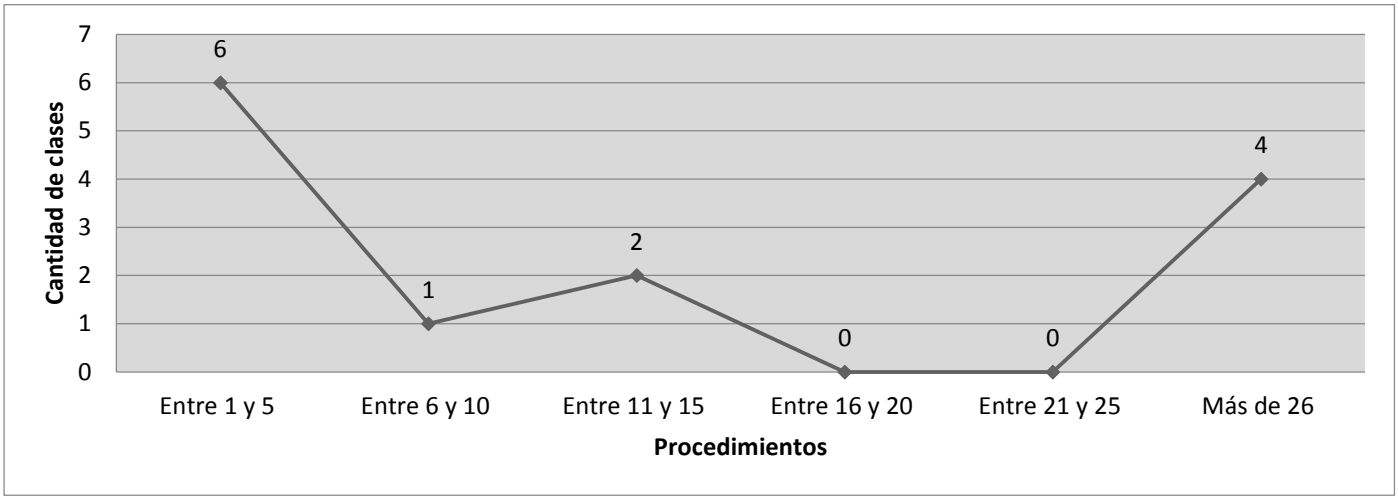


Figura 13: Representación de la métrica TOC.

La Figura 14 muestra los resultados obtenidos en porciento agrupados en los intervalos definidos.

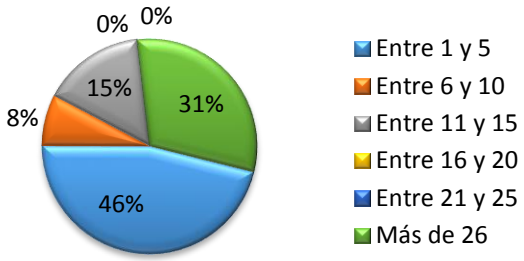


Figura 14: Representación en porciento de la evaluación de la métrica TOC.

La Figura 15 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica TOC en el atributo responsabilidad, demostrando un resultado satisfactorio, ya que el 69% de las clases tienen una baja responsabilidad. Esta característica, al estar distribuida de forma equilibrada, permite que en caso de fallos, ninguna clase sea demasiado crítica como para dejar al sistema fuera de servicio.

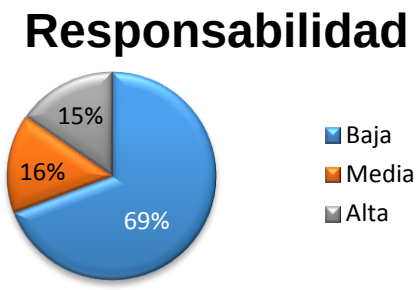


Figura 15: Representación de la evaluación de la métrica TOC en el atributo responsabilidad.

La Figura 16 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica TOC, en el atributo complejidad de implementación, demostrando un resultado satisfactorio ya que la mayoría de las clases tienen una baja complejidad.

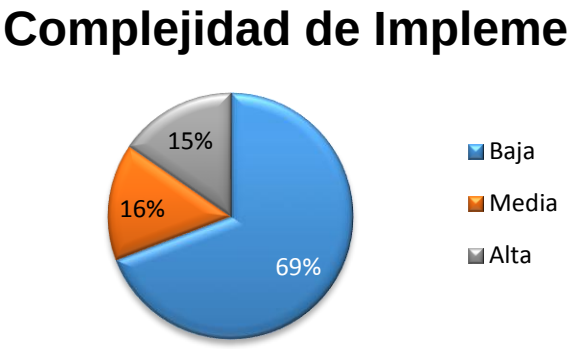


Figura 16: Representación de la evaluación de la métrica TOC en el atributo complejidad de implementación.

La Figura 17 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica TOC, en el atributo reutilización, demostrando que el diseño tiene un grado de eficiencia aceptable ya que solamente 15% de las clases poseen una baja reutilización.

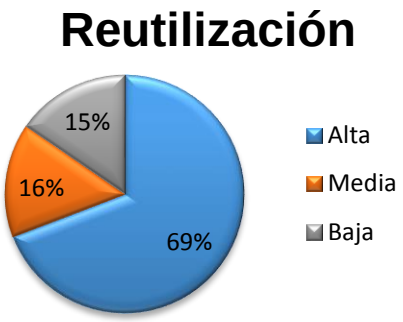


Figura 17: Representación de la evaluación de la métrica TOC en el atributo reutilización.

Analizando los resultados obtenidos de la métrica TOC, se puede concluir que el diseño tiene una calidad aceptable, teniendo en cuenta los resultados arrojados por los atributos analizados. Solo el 15% de las clases presentan una alta responsabilidad, una alta complejidad de implementación y una baja reutilización, lo cual demuestra que el resultado es satisfactorio.

Relaciones entre Clases (RC): está dado por el número relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- **Acoplamiento:** Un aumento del RC implica un aumento del Acoplamiento de la clase.
- **Complejidad de mantenimiento:** Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** Un aumento del RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

La Tabla 7 muestra el cálculo de los atributos de calidad.

Tabla 7: Cálculo de los atributos de calidad.

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

La Figura 18 muestra los resultados obtenidos en porciento agrupados en los intervalos definidos. Se demuestra que el 62% de las clases tienen cero dependencias y el 15% tiene una dependencia con otra clase, arrojando un resultado positivo ya que demuestra el 92% de las clases tienen un nivel aceptable de la calidad.

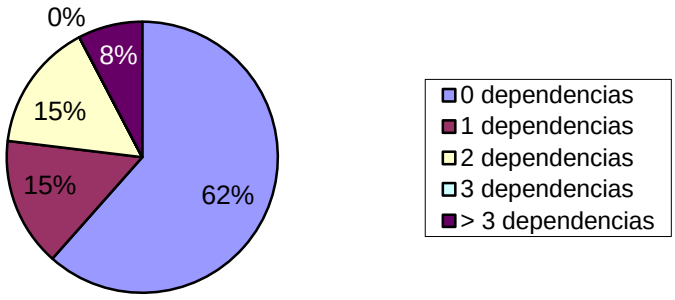


Figura 18: Representación en porciento de la evaluación de la métrica RC.

La Figura 19 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica RC, en el atributo acoplamiento, demostrando que se evidencia un bajo acoplamiento entre las clases, ya que el 62% de las clases no presentan dependencia con otra y el 15% una dependencia con otra. El resultado es positivo ya que al existir poca dependencia entre las clases, aumenta el grado de reutilización.

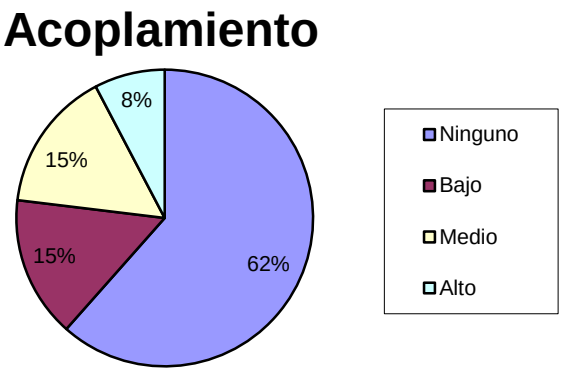


Figura 19: Representación de la evaluación de la métrica RC en el atributo acoplamiento.

La Figura 20 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica RC, en el atributo complejidad de mantenimiento, demostrando un resultado aceptable, ya que el 77% de las clases presentan baja complejidad de mantenimiento.

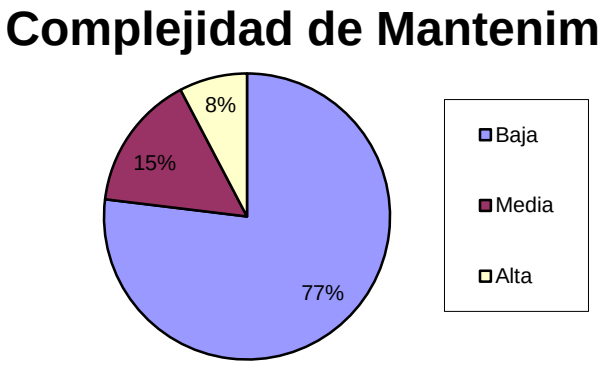


Figura 20: Representación de la evaluación de la métrica RC en el atributo complejidad de mantenimiento.

La Figura 21 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica RC, en el atributo cantidad de pruebas.

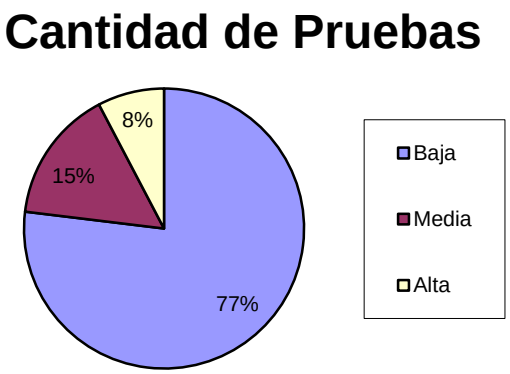


Figura 21: Representación de la evaluación de la métrica RC en el atributo cantidad de pruebas.

La Figura 22 refleja la representación de la incidencia de los resultados obtenidos de la evaluación de la métrica RC, en el atributo reutilización.

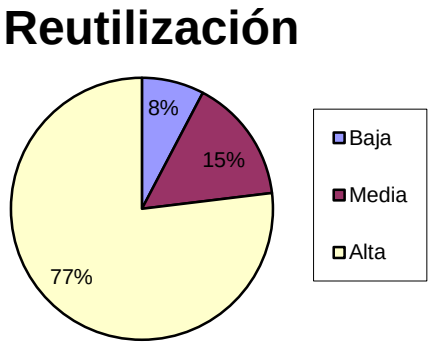


Figura 22: Representación de la evaluación de la métrica RC en el atributo reutilización.

Analizando los resultados obtenidos de la métrica RC, se puede concluir que el diseño tiene una calidad aceptable, teniendo en cuenta los resultados arrojados por los atributos analizados. El 92% de las clases poseen menos de tres dependencias con otras clases. Los atributos de calidad se encuentran en un nivel satisfactorio, ya que el 77% de las clases el nivel de acoplamiento es mínimo, la cantidad de pruebas y la complejidad de mantenimiento son bajas con un 77% y un alto grado de reutilización al 77%.

2.7 Implementación

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 para facilitar la lectura, comprensión y mantenimiento del código. Estos establecerán las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que pueda aumentar su mantenibilidad a lo largo del tiempo. (CEIGE, 2012) A continuación se especifican los estándares de codificación que fueron utilizados en el desarrollo de la Herramienta para la generación de consultas de Doctrine:

Nomenclatura de las clases:

- Para las clases controladoras el nombre comenzará con la primera letra en mayúscula y el resto en minúscula, además estará dado según la función que realiza y se le adicionará al final "**Controller**". Ejemplo: **QgeneratorController**.
- El nombre de las clases del modelo que se encuentran dentro de la carpeta "**bussines**" comenzará con la primera letra en mayúscula y el resto en minúscula. A este nombre se le agregará al final "**Model**". Ejemplo: **ClausulagroupbyMode**, **ClausulawhereModel**.
- El nombre de las clases de la vista comenzará con la primera letra en mayúscula y el resto en minúscula. Ejemplo: **Panelsuperior**, **Panelinferior**.

Nomenclatura de las funciones:

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing**, y con sólo leerlo se reconoce el propósito de la misma. En caso de ser una acción de la clase controladora se le pone el nombre y seguida la palabra:"Action". Ejemplo: **descargarFicheroAction()**, **eliminarTodos()**.

Nomenclatura de las variables:

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing*,* y comenzando con un prefijo según el tipo de datos. Ejemplo: **btnGuardar**, **storeEntidad**.

Nomenclatura de los atributos:

El nombre a emplear para los atributos se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing**. Además en caso de ser un objeto se comienza con:"_" y después se escribe el nombre. Ejemplo: objProject=_project.

2.8 Interfaz de usuario

A continuación se describe en las Figuras 23, 24 y 25, las funcionalidades de la herramienta generadora de consultas DQL para facilitar el trabajo de los programadores del Departamento de Desarrollo de Componentes.

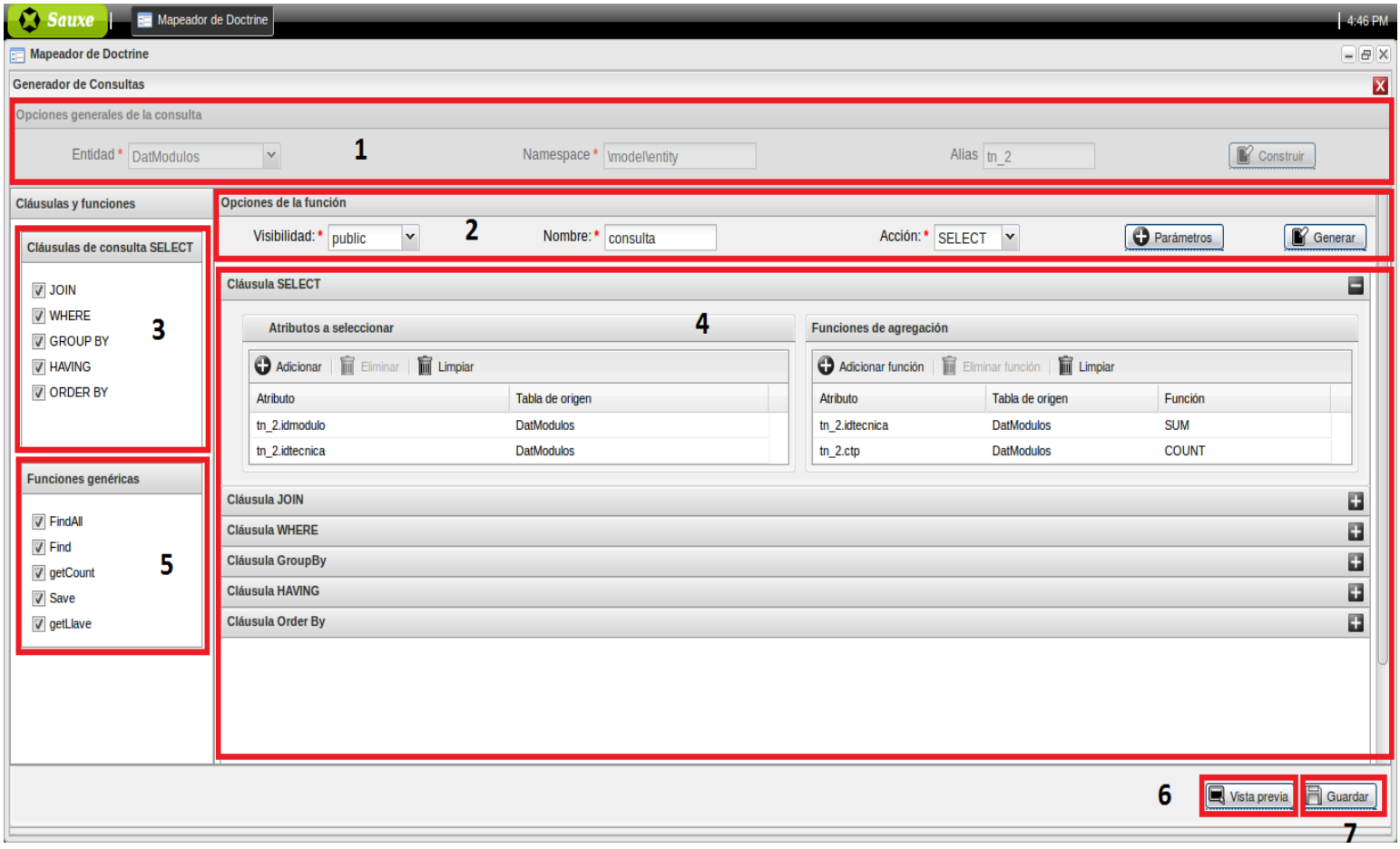


Figura 23: Interfaz de usuario principal.

1. El *panel superior* permite escoger la entidad a la cual se le agregará, las funciones que contienen las consultas DQL y esta generará automáticamente el alias de la entidad, además de especificar el namespace de la entidad. Después se adicionan todos los datos al presionar el botón “Construir” que habilitará el panel de opciones de funciones y las funciones genéricas.
2. El panel de *opciones de funciones* permite crear las funciones con su respectivo nombre, visibilidad, acción y parámetros. Al seleccionar la acción deseada se habilitará el panel de cláusulas. El usuario después de haber realizado la consulta deseada presiona el botón “Generar” y este permitirá ir conformando la clase con las funcionalidades específicas.
3. En el *panel de cláusulas* se puede agregar las cláusulas necesarias para conformar la consulta.
4. Se muestra las *cláusulas* donde se adicionarán los datos necesarios para realizar la consulta.
5. Las *funciones autogeneradas* permiten agregar al código una función predefinida.
6. La “Vista previa” muestra las funciones que el usuario ha creado.
7. La opción “Guardar” crea un fichero php con las funciones creadas.

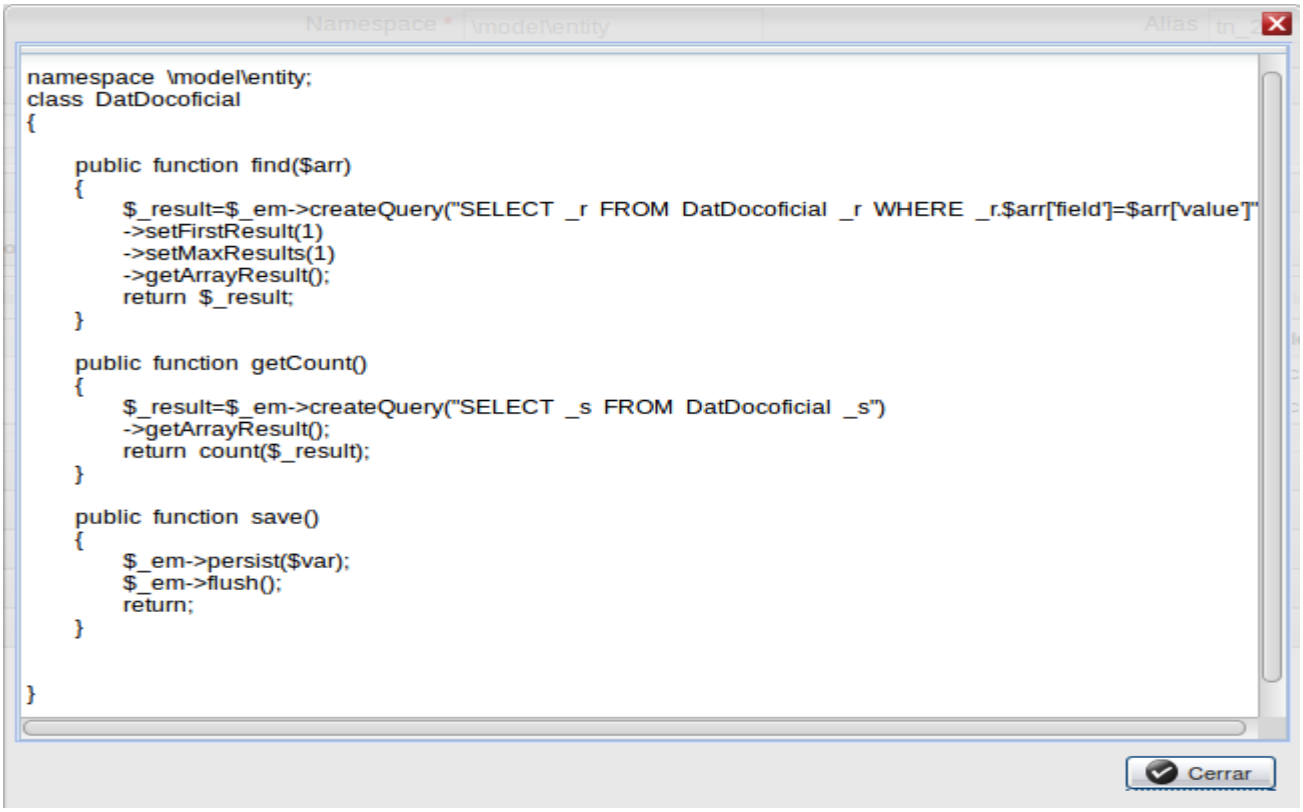


Figura 24: Vista previa del código.

```php
<?php

namespace \models\Entity
class Carrera
{

    public function obtener($_em){
        $consulta = 'SELECT tn_1.idusuario, tn_1.fecha
        FROM \models\Entity\Carrera tn_1 WHERE tn_1.estado = tn_1.descripcion';
        $query = $_em->createQuery($consulta);
        $resultado = $query->getResult();
        return $resultado;
    }

    public function eliminar($_em,$idusuario){
        $consulta = 'DELETE \models\Entity\Carrera tn_1 WHERE tn_1.idusuario = :idusuario';
        $query = $_em->createQuery($consulta);
        $query->setParameter(array(
        'idusuario' => '3',
        ));
        $resultado = $query->getResult();
        return $resultado;
    }
}
```

Figura 25: Fichero php generado.

Conclusiones parciales

El presente capítulo permitió obtener un mayor conocimiento de la herramienta en creación; en él se definió el modelo conceptual, que contribuyó a una mejor compresión de los conceptos utilizados en su desarrollo. Con la descripción de los requisitos funcionales y no funcionales se pudieron determinar las principales funcionalidades que brindará la herramienta, también se creó el diagrama de clases del diseño, el cual brinda información específica sobre la solución. Además, se confeccionaron las historias de usuarios, las cuales brindarán más explícita las funcionalidades de cada uno de los requisitos. Dado lo descrito anteriormente, se da cumplimiento al desarrollo del análisis y diseño relacionado con la herramienta de generación de código DQL. Una vez concluido el diseño e implementación de la solución puede darse paso a la validación de la solución.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

La realización de pruebas es una de las fases indispensables para validar la calidad y correcto funcionamiento de un software. La misma es el proceso de ejecución de una aplicación, con el propósito de comprobar que el producto satisfaga los requerimientos específicos, y que tenga el resultado esperado. A la herramienta generadora de consultas DQL, se hicieron varias pruebas para validar la calidad de la misma, entre ellas se encuentran las pruebas de caja blanca, caja negra y prueba de aceptación, las cuales se explican en el presente capítulo.

3.1 Pruebas de software

Las pruebas de software son actividades que se realizan, en las cuales se utilizan diseños de prueba, el artefacto es ejecutado o evaluado bajo requisitos especificados y se registran los resultados obtenidos, los que deben ser comparados con los resultados esperados, para emitir una evaluación basada en aspectos determinados del artefacto. (Capote García, 2011)

3.1.1 Pruebas de caja blanca

Las pruebas de caja blanca comprueban los caminos lógicos del software, proponiendo casos de prueba que se ejerciten de conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado. Estas pruebas requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. (Menéndez Ávalos, 2012)

La prueba del **camino básico** es una técnica de prueba de caja blanca propuesta por Tom McCabe, esta se emplea en la solución, donde es necesario el cálculo de la complejidad ciclomática del algoritmo a analizar, para lo cual se deben enumerar las sentencias de código del mismo y a partir de ello elaborar el grafo de flujo de esta funcionalidad. (Pressman R. , 2010) A continuación las Figura 26 enumera las sentencias de código del método **eliminarAtributosSelect()**.

```php
public function eliminarAtributoSelect($id){
    $i =0; $pos = NULL;//1
    foreach ($this->atributos as $atrib) {//2
        if ($atrib['id'] == $id){//3
            $pos = $i;//4
            unset($this->atributos[$pos]);//4
            $this->atributos = array_values($this->atributos);//4
            break;//4
        }
        $i++;//5
    }
}//6
```

Figura 26: Código fuente del método eliminarAtributosSelect().

La Figura 27 muestra el grafo del flujo asociado al método **eliminarAtributosSelect()**.

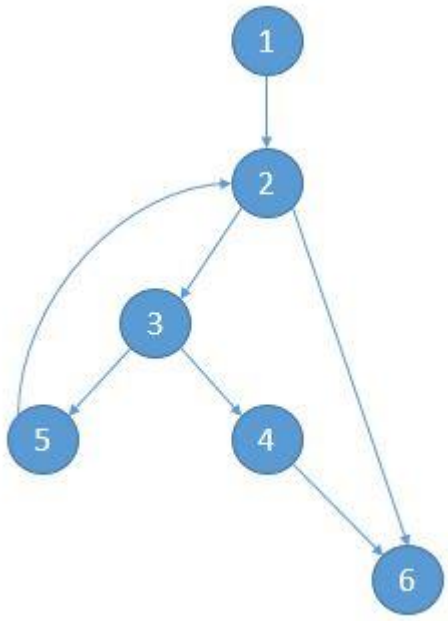


Figura 27: Grafo de flujo asociado al método eliminarAtributosSelect().

La complejidad ciclomática es una métrica de software extremadamente útil, pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática, define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior

para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Después de haber realizado el grafo, se calcula la complejidad ciclomática por tres fórmulas distintas, las cuales deben dar el mismo resultado para comprobar que el cálculo sea correcto.

La complejidad ciclomática, V (G) se define como:

1. **V (G) = R** donde: **R** representa la cantidad total de regiones del grafo.
   V (G) = 3

2. **V (G) = A – N + 2** donde: **A** es el número de aristas del grafo y **N** es el número de nodos.
   V (G) = 7 – 6 + 2
   V (G) = 3

3. **V (G) = P + 1** donde: **P** es el número de nodos predicados contenidos en el grafo.
   V (G) = 2 + 1
   V (G) = 3

El cálculo de las diferentes fórmulas anteriormente nombradas arrojó el mismo resultado, por esto se puede afirmar que la complejidad ciclomática del método es tres. Esto significa que existen tres posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

- **Camino básico #1: 1 – 2 – 3 – 4 – 6**
- **Camino básico #2: 1 – 2 – 3 – 5 – 2 – 3 – 4 – 6**
- **Camino básico #3: 1 – 2 – 6**

Caso de prueba para el camino básico # 1

Camino: 1 – 2 – 3 – 4 – 6
Descripción: El dato se obtiene a través del parámetro id.
Entrada: $id = 1.
Resultados esperados: Se espera que elimine el atributo de la lista de atributos de la cláusula Select y muestre un mensaje informando que el atributo fue eliminado correctamente.

Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 2

Camino: 1 – 2 – 3 – 5 – 2 – 3 – 4 – 6
Descripción: El dato se obtiene a través del parámetro id.
Entrada: $id = 2.
Resultados esperados: Se espera que elimine el atributo de la lista de atributos de la cláusula Select y muestre un mensaje informando que el atributo fue eliminado correctamente.
Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 3

Camino: 1 – 2 – 6
Descripción: El dato se obtiene a través del parámetro id.
Entrada: $id = 0.
Resultados esperados: Se espera que no se elimine el atributo de la lista de atributos de la cláusula Select porque no existen atributos en la lista.
Resultados obtenidos: Satisfactorio.

Con la realización de las pruebas de caja blanca al método ***eliminarAtributoSelect(),*** se obtuvieron los resultados esperados, debido a que las tres fórmulas explicadas arrojaron el mismo resultado y se cumplió la ejecución de cada camino básico al menos una vez con los parámetros de entrada y las salidas esperadas.

3.1.2 Pruebas de caja negra

Pruebas que se realizan sobre la interfaz del software. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, y que la integridad de la información externa se mantiene. (Echeverría Perez, 2011)
Para evaluar los requisitos funcionales de la herramienta generadora de consultas de Doctrine se realizaron pruebas de caja negra en la fase de pruebas internas. Para aplicar estas pruebas se conformaron los diseños de casos de pruebas correspondientes a cada una de las funcionalidades de la herramienta. A continuación la Tabla 6 muestra un ejemplo del caso de prueba “Adicionar atributo”.

Tabla 8: Caso de prueba "Adicionar atributo".

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Adicionar atributos introduciendo los datos.	*Se adiciona atributos en la cláusula Select.*	*Se adiciona el atributo a la cláusula Select.*	*- Se selecciona la opción "Adicionar" en la cláusula Select. - El programador llena los campos. - Se presiona el botón Aceptar. - Si los campos están llenos se Adiciona el atributo.*
EC 1.2 Adicionar atributos dejando campos vacíos.	*Se adiciona atributos en la cláusula Select dejando los campos vacíos.*	*Se muestra el mensaje "Existen campos vacíos".*	*- Se selecciona la opción "Adicionar" en la cláusula Select. - El programador deja campos vacíos. - Se presiona el botón Aceptar - Se muestra un mensaje de error.*
EC 1.2 Aplicar.	*Se presiona el botón Aplicar.*	*Se adiciona el atributo a la cláusula Select.*	*- Se selecciona la opción "Adicionar" en la cláusula Select. - El programador llena los campos. - Se presiona el botón Aplicar. - Se presiona el botón Aceptar - Si los campos están llenos se adiciona el atributo.*
EC 1.2 Cancelar	*Se presiona el botón Cancelar.*	*Se cierra la ventana.*	*- Se selecciona la opción "Adicionar" en la cláusula Select. - El programador llena los campos o no. - Se presiona el botón Cancelar.*

Una vez realizado todos los casos de pruebas, se recogen las nos conformidades detectadas por el Departamento de Desarrollo de Componentes. Para verificar la calidad de la herramienta se realizaron tres iteraciones, las cuales arrojaron varias no conformidades que se muestran en la Figura 28:

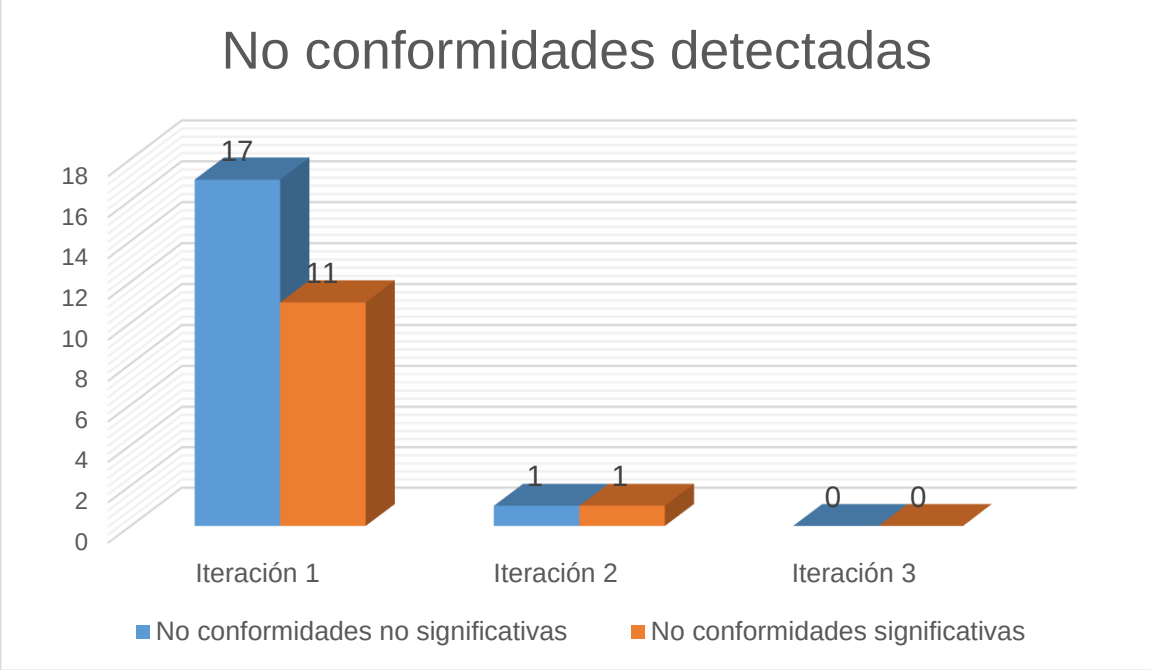


Figura 28: No conformidades detectadas en la aplicación.

3.1.3 Comparación de generación de consultas

Se realizaron pruebas de comparación entre funciones DQL de las clases existentes en el repositorio del Departamento de Desarrollo de Componente y las creadas por la herramienta generadora de consultas DQL, analizando su sintaxis y composición.

La función de comparación creada de forma manual por los desarrolladores del Departamento de Desarrollo de Componentes, fue obtenida del sistema de seguridad Acaxia, de la clase “DatSistema”. La Figura 29 muestra la función “obtenerSistemaP”.

```
public function obtenerSistemasP($_em)
    {
        $result = $_em
            ->createQuery('select a.denominacion, a.abreviatura
            from seguridad\models\Entity\DatSistema a where a.idpadre = a.idsistema ')
            ->getResult();
        return $result;
    }
```

Figura 29: Función DQL de comparación.

A continuación se muestra la Figura 30, que contiene la función “obtenerSistemaP” creada a partir de la Herramienta para la generación de consultas de Doctrine, dicha función se elaboró a partir de la misma clase “DatSistema” del sistema de seguridad Acaxia.

```
public function obtenerSistemasP($_em)
{
    $consulta = "SELECT tn_1.denominacion, tn_1.abreviatura
    FROM \seguridad\model\Entity\DatSistema tn_1 WHERE tn_1.idpadre = tn_1.idsistema";
    $query = $_em->createQuery($consulta);
    $resultado = $query->getResult();
    return $resultado;
}
```

Figura 30: Función DQL generada por la herramienta.

Esta comparación arrojó como resultado positivo que las funciones comparadas eran similares y devolvían el mismo resultado, ya que presentan la sintaxis correcta en Doctrine v2. En ellas se evidencian una correcta utilización de las palabras reservadas como el “Select” y el “Where”. Además la función generada por la herramienta esta mejor estructurada facilitando a los programadores comprender el código y elimina los posibles errores de sintaxis.

3.2 Resultado en eventos

La herramienta obtuvo premio relevante en la Jornada del Ingeniero en Ciencias Informáticas (JICI) a nivel de facultad, destacado en la 10ma Peña Tecnológica, en la XXVII Jornada Científica Estudiantil Nacional realizada en el ISMI Eliseo Reyes Rodríguez “Capitán San Luis” y en la XIII Jornada Científica Estudiantil a nivel de facultad, además de mención en la JICI a nivel UCI. La evidencia de los resultados antes mencionados se puede ver en los anexos del 6 al 10.

3.3 Pre-experimento

Con el objetivo de conocer si se obtiene una disminución del tiempo de creación de las funciones haciendo uso de la Herramienta para la generación de consultas de Doctrine respecto a elaboración manual de las mismas, se procede a realizar un Pre-experimento. A continuación se muestran los resultados del método empleado.

Se seleccionó un grupo de especialistas del Departamento de Desarrollo de Componentes entre ellos se encuentra un máster en ciencias y tres ingenieros los cuales hacen uso de Doctrine v2 en el desarrollo de aplicaciones y con al menos un año de experiencia en esta tecnología, estos se muestran en la Tabla 9.

Tabla 9: Especialistas seleccionados del Departamento de Desarrollo de Componentes.

Nombre y Apellidos	Categoría	Experiencia de trabajo (años)	Experiencia con Doctrine v2 (años)
Ing.Inoelkis Velazquez Osorio	Ingeniero	3	3
Ing.Katia Saria Preval	Ingeniero	2	1
MSc.Orlando Valenzuela Aguilera	Máster en Ciencias	8	1
Ing.Claudia Bravo Batista	Ingeniero	3	1

Los especialistas realizaron cinco funciones de forma manual, de estas una es de complejidad baja, dos de complejidad media y dos de complejidad alta. En la Tabla 10 se muestra los resultados obtenidos en minutos de estas funciones.

Tabla 10: Resultados obtenidos en minutos de la creación de las funciones.

Nombre y Apellidos	Básica	Media	Alta	Total
Ing.Inoelkis Velazquez Osorio	2	6	15	**23**
Ing.Katia Saria Preval	3	9	20	**32**

MSc.Orlando Valenzuela Aguilera	4	10	18	**32**
Ing.Claudia Bravo Batista	6	15	25	**46**
Total	**15**	**40**	**78**	**133**

Después de haber realizado las funciones de forma manual, se realizaron las mismas mediante la herramienta generadora de consultas DQL, obteniendo los siguientes resultados en minutos que se muestran en la Tabla 11.

Tabla 11: Resultados obtenidos de la creación de las funciones mediante la herramienta generadora de consultas DQL.

Nombre y Apellidos	Básica	Media	Alta	Total
Ing.Inoelkis Velazquez Osorio	1	4	11	**16**
Ing.Katia Saria Preval	1	6	15	**22**
MSc.Orlando Valenzuela Aguilera	1	7	12	**20**
Ing.Claudia Bravo Batista	1	5	15	**21**
Total	**4**	**22**	**53**	**79**

Una vez realizado el pre-experimento se obtuvieron los siguientes resultados (Figura 31):

- ✓ El tiempo necesario para la elaboración de las funciones de forma manual fue de 133 minutos.
- ✓ El tiempo necesario para la elaboración de las funciones haciendo uso de la herramienta generador de consultas fue de 79 minutos.

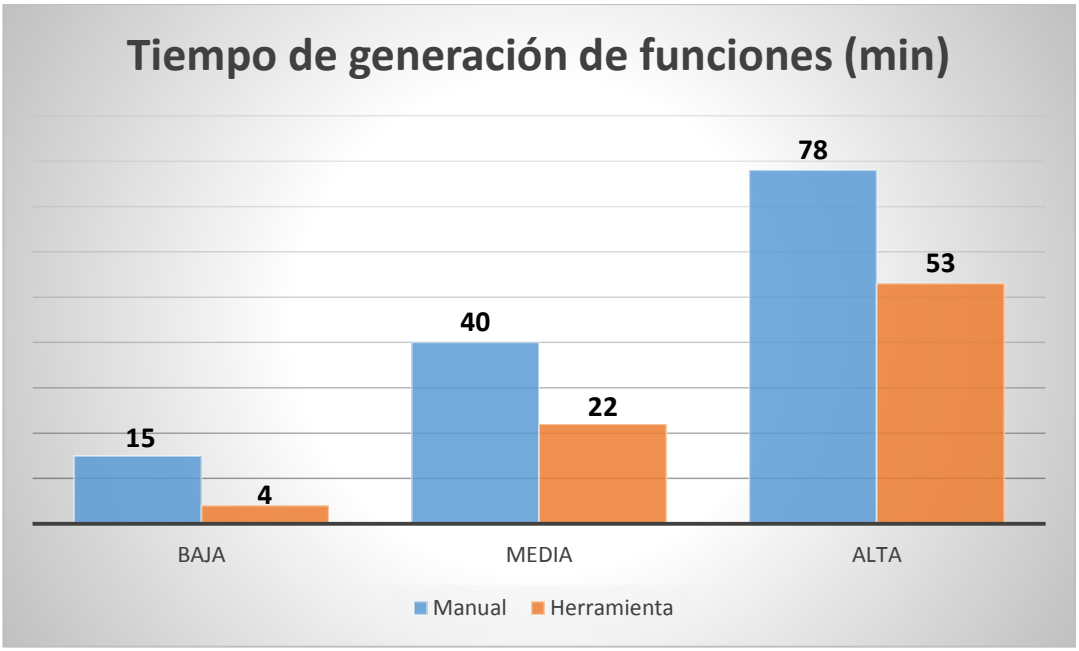


Figura 31: Resultado del pre-experimento.

A partir de los resultados obtenidos se llega a la conclusión, que la utilización de la herramienta evidenció una disminución de 54 minutos, esto representa el 41% del tiempo total necesario a la hora de realizar las funciones de forma manual.

Conclusiones Parciales

Del proceso de pruebas de software al que fue sometido la herramienta, se obtuvo como resultado la validez de las estructuras internas a través de las pruebas de caja blanca, así como pruebas funcionales las cuales arrojaron resultados satisfactorios a través de una liberación funcional.

Con la pruebas de comparación de generación consultas se llegó a un resultado satisfactorio ya que se evidenció que la sintaxis y la estructura de las funciones generadas fueron las correctas.

El pre-experimento demostró que la herramienta disminuye el tiempo de implementación de la capa de acceso a datos de las aplicaciones desarrolladas empleando Doctrine.

CONCLUSIONES

Luego del desarrollo de la Herramienta para la generación de consulta de Doctrine, se arribaron a las siguientes conclusiones:

- Se realizó un estudio del estado del arte sobre las herramientas generadoras de consultas existentes en la actualidad, donde se obtuvo como resultado, que estas no generan código DQL.
- Se realizó el análisis y diseño de la herramienta, el cual fue validado mediante las métricas Tamaño Operacional de Clase y Relaciones entre Clases arrojando como resultados que la herramienta tiene alta reutilización, bajo acoplamiento, baja complejidad de implementación y un fácil mantenimiento.
- Se desarrolló la Herramienta para la generación de consulta de Doctrine, empleando las herramientas y tecnologías, definidas por el Departamento de Desarrollo de Componente, obteniendo un producto funcional acorde a los requisitos identificados.
- La herramienta fue probada por cuatro especialistas del Departamento de Desarrollo de Componentes, los cuales comprobaron que la utilización de la herramienta evidenció una disminución del tiempo en la creación de las consultas y cumple con los objetivos determinados en la investigación.

RECOMENDACIONES

- Incluir en la herramienta una nueva funcionalidad para la creación de consultas anidadas, la cual se utiliza para crear consultas dentro de otras consultas.
- Incluir un diseñador gráfico de consultas DQL para mejorar la usabilidad de la herramienta hacia los usuarios finales.

REFERENCIA BIBLIOGRÁFICA

(2014). Recuperado el 9 de febrero de 2015, de pgAdmin:PostgresSQL: http://www.pgadmin.org/

(9 de febrero de 2015). Obtenido de EMS SQL MANAGER: http://www.sqlmanager.net

Bahit, E. (2011). *MVC con PHP.*

Baryolo, O. G. (2010). SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA. *Tesis de Maestría*. La Habana, Universidad de las Ciencias Informáticas.

Camps Riba, J. M. (2012). Diseño de un framework de persistencia. *Tesis de Maestría*. España: Universitat Oberta de Catalunya.

Capote García, T. (2011). Conceptualización e implantación de un Laboratorio Industrial de Pruebas de Software. *Tesis de Maestría*. La Habana, Universidad de las Ciencias Infomáticas.

CEIGE. (2012). *Estándares de codificación Proyectos con el marco de trabajo Sauxe del CEIGE.*

Cillero, M. (2014). *manuel.cillero.* Obtenido de http://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-clases

Codisa. (2014). Recuperado el 17 de Enero de 2015, de http://www.codisa.com/herramientas-query-builder.html

Echeverría Perez, D. (2011). Desarrollo de una ontología de apoyo al procedimiento del Departamento de Pruebas de Software. *Tesis de Maestría*. La Habana, Universidad de las Ciencias Informáticas.

Foundation, T. (2012). Recuperado el 16 de Enero de 2015, de The Apache Software Foundation: http://www.apache.org

Ganesh, G. S. (2008). *Requirements Engineering: Elicitation Techniques.* University West, Departament of Technology, Mathematics and Computer Science, Suecia.

Guerrero, C., Londoño, J., Suárez, J., & Gutiérrez, L. (2014). ESTUDIO COMPARATIVO DE MARCOS DE TRABAJO PARA EL DESARROLLO SOFTWARE ORIENTADO A ASPECTOS. *Información Tecnológica, 25*(2). Recuperado el 17 de Enero de 2015, de Scielo: http://www.scielo.cl/scielo.php?pid=S0718-07642014000200008&script=sci_arttext

IEEE. (2007). Introducción a la Ingeniería de Requisitos.

INTECO. (2010). *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* España: Laboratorio Nacional de Calidad del Software.

Larman, C. (2004). *UML y Patrones.*

Menéndez Ávalos, H. (2012). Procedimiento para el diseño e implantación de pruebas a partir del análisis de los riesgos del producto. *Tesis de Maestría*. La Habana, Universidad de las Ciencias Informáticas.

Minnetto, E. (2007). *Frameworks para desenvolvimiento en php.* Brasil, Novatec.

NetBeans. (2014). Recuperado el 17 de Enero de 2015, de netbeans.org

Pacheco, N. (2011). *Doctrine 2 ORM.* Recuperado el 17 de Enero de 2015

PHP. (2014). Recuperado el 17 de Enero de 2015, de Manual de PHP: http://php.net/manual/en/index.php

PMBOK. (2013). *Guía de los Fundamentos para la Dirección de Proyectos 5ta Edicion .*

PMOinformatica. (2014). *La oficina de proyectos de informática*. Obtenido de http://www.pmoinformatica.com/2013/04/que-son-las-historias-de-usuario-7.html

PostgresSQL. (2013). Recuperado el 17 de Enero de 2015, de PostgresSQL: http://www.postgresql.org/

Pressman, R. (2005). *Ingeniería del Softwre: Un enfoque práctico.* Quinta edición.

Pressman, R. (2010). *Software Engineering: A Practitioner´s Approach.* Seventh Edition.

Pupo, Y. (2010). *Libro de Ayuda del Marco de Trabajo Sauxe.*

Quintana Aput, R. (2014). El diseño metodológico de la Investigación Científica., (pág. 63).

Rodríguez Sánchez, T. (2014). *Metodología de desarrollo para la Actividad productiva de la UCI.*

Sánchez Rosas, J. (2008). *Desarrollo en Web.* Obtenido de http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/

Sencha inc. (2014). Recuperado el 17 de Enero de 2015, de Sencha: http://www.sencha.com/products/extjs/#overview

Server Gove. (2010). *Doctrine.* Obtenido de http://www.doctrine-project.org/

Sommerville, I. (2007). *Software Engineering* (Eighth Edition ed.).

Stellman, A., & Greene, J. (2005). *Applied Software Project Management.* EEUU: O'Really Media.

System, M. (2014). Recuperado el 17 de Enero de 2015, de http://www.multisystem.cl/productos/aquafold2/generador-visual-de-consultas/

Thayer, R., & Merlin, D. (1997). *IEEE Software Requirement Engineering Second Edition.* New York.

UML. (2014). Recuperado el 17 de Enero de 2015, de UML: www.uml.org

Visual Paradigm. (2014). Recuperado el 17 de Enero de 2015, de www.visual-paradigm.com

ANEXOS

Anexo 1. Historia de usuario adicionar función de agregación.

Tabla 12: Historia de usuario adicionar función de agregación.

Historia de Usuario	
Número:HU8.1	**Nombre del requisito:** Adicionar función de agregación.
Programador: Gabriel Ibarra García.	**Iteración Asignada:** Primera
Prioridad: Media.	**Tiempo Estimado:** Siete días.
Riesgo en Desarrollo: Media.	**Tiempo Real:** Una semana.
Descripción: La historia de usuario permite adicionar funciones de agregación en los atributos de la cláusula select.	
Observaciones:	
Prototipo de interfaz: 	

Anexo 2. Historia de usuario adicionar atributo.

Tabla 13: Historia de usuario adicionar atributo.

Historia de Usuario	
Número: HU7.1	**Nombre del requisito:** Adicionar atributos.

Programador: Gabriel Ibarra García.	**Iteración Asignada:** Primera
Prioridad: Media.	**Tiempo Estimado:** Siete días.
Riesgo en Desarrollo: Media.	**Tiempo Real:** Una semana.
Descripción: La historia de usuario permite adicionar atributos en la cláusula select.	
Observaciones:	
Prototipo de interfaz: 	

Anexo 3. Instrumento de evaluación de las métricas TOC y RC.

Tabla 14: Instrumento de evaluación de la métrica TOC.

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
GestaccionController	53	Alta	Alta	Baja
Clausulafrom	5	Baja	Baja	Alta
ClausulaGroupby	5	Baja	Baja	Alta
Clausulahaving	12	Baja	Baja	Alta
Clausulaorderby	5	Baja	Baja	Alta
ClausulaSelect	9	Baja	Baja	Alta
Clausulaset	5	Baja	Baja	Alta
Clausulawhere	13	Baja	Baja	Alta
Consultadql	35	Media	Media	Media
Funcion	44	Alta	Alta	Baja

Parametro	5	Baja	Baja	Alta
Query	34	Media	Media	Media
Storev2	5	Baja	Baja	Alta

Tabla 15: Instrumento de evaluación de la métrica RC.

Clase	Cantidad de Relaciones	Acoplamiento	Complejidad de Mantenimiento	Reutilización	Cantidad de Pruebas
GestaccionController	1	Bajo	Baja	Alta	Baja
Clausulafrom	0	Ninguno	Baja	Alta	Baja
ClausulaGroupby	0	Ninguno	Baja	Alta	Baja
Clausulahaving	0	Ninguno	Baja	Alta	Baja
Clausulaorderby	0	Ninguno	Baja	Alta	Baja
ClausulaSelect	0	Ninguno	Baja	Alta	Baja
Clausulaset	0	Ninguno	Baja	Alta	Baja
Clausulawhere	0	Ninguno	Baja	Alta	Baja
Consultadql	2	Medio	Media	Media	Media
Funcion	2	Medio	Media	Media	Media
Parametro	1	Bajo	Baja	Alta	Baja
Query	8	Alto	Alta	Baja	Alta
Storev2	0	Ninguno	Baja	Alta	Baja

Anexo 4. Caso de prueba Adicionar función de agregación.

Tabla 16: Caso de prueba Adicionar función de agregación.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Adicionar función de agregación introduciendo los datos.	Se adiciona la función de agregación en la cláusula select.	Se adiciona la función de agregación a la cláusula select.	- Se selecciona la opción "Adicionar función" en la cláusula select. - El programador llena los campos. - Se presiona el botón Aceptar - Si los campos están llenos se agrega la función de agregación.

EC 1.2 Adicionar función de agregación dejando campos vacíos.	Se adiciona la función de agregación en la cláusula select dejando los campos vacíos.	Se muestra el mensaje de error “Existen campos vacíos”.	- Se selecciona la opción "Adicionar función" en la cláusula select. - El programador deja campos vacío. - Se presiona el botón Aceptar - Se muestra un mensaje de error.
EC 1.3 Aplicar.	Se presiona el botón Aplicar.	Se adiciona el atributo a la cláusula select.	- Se selecciona la opción "Adicionar función" en la cláusula select. - El programador llena los campos. - Se presiona el botón Aplicar. - Se presiona el botón Aceptar - Si los campos están llenos se adiciona el atributo.
EC 1.4 Cancelar	Se presiona el botón Cancelar.	Se cierra la ventana.	- Se selecciona la opción "Adicionar función" en la cláusula select. - El programador llena los campos o no. - Se presiona el botón Cancelar.

Anexo 5. Caso de prueba Adicionar atributo de ordenación.

Tabla 17: Caso de prueba Adicionar atributo de ordenación.

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Adicionar atributos de ordenación introduciendo los datos.	Se adiciona atributos de ordenación en la cláusula order by.	Se adiciona el atributo de ordenación a la cláusula order by.	- Se selecciona la opción "Adicionar" en la cláusula order by. - El programador llena los campos. - Se presiona el botón Aceptar - Si los campos están llenos se adiciona el atributo.

EC 1.2 Adicionar atributos de ordenación dejando campos vacíos.	Se adiciona atributos de ordenación en la cláusula order by dejando los campos vacíos.	Se muestra el mensaje de error “Existen campos vacíos”.	- Se selecciona la opción "Adicionar" en la cláusula order by. - El programador deja campos vacío. - Se presiona el botón Aceptar - Se muestra un mensaje de error.
EC 1.3 Aplicar.	Se presiona el botón Aplicar.	Se adiciona el atributo a la cláusula order by.	- Se selecciona la opción "Adicionar" en la cláusula order by. - El programador llena los campos. - Se presiona el botón Aplicar. - Se presiona el botón Aceptar - Si los campos están llenos se adiciona el atributo.
EC 1.4 Cancelar	Se presiona el botón Cancelar.	Se cierra la ventana.	- Se selecciona la opción "Adicionar" en la cláusula order by. - El programador llena los campos o no. - Se presiona el botón Cancelar.

Anexo 6. Certificado de la 10 Peña Tecnológica.

Figura 32: Certificado de la 10 Peña Tecnológica.

Anexo 7. Certificado de la JICI a nivel de facultad.

Figura 33: Certificado de la JICI a nivel de facultad (Gabriel).

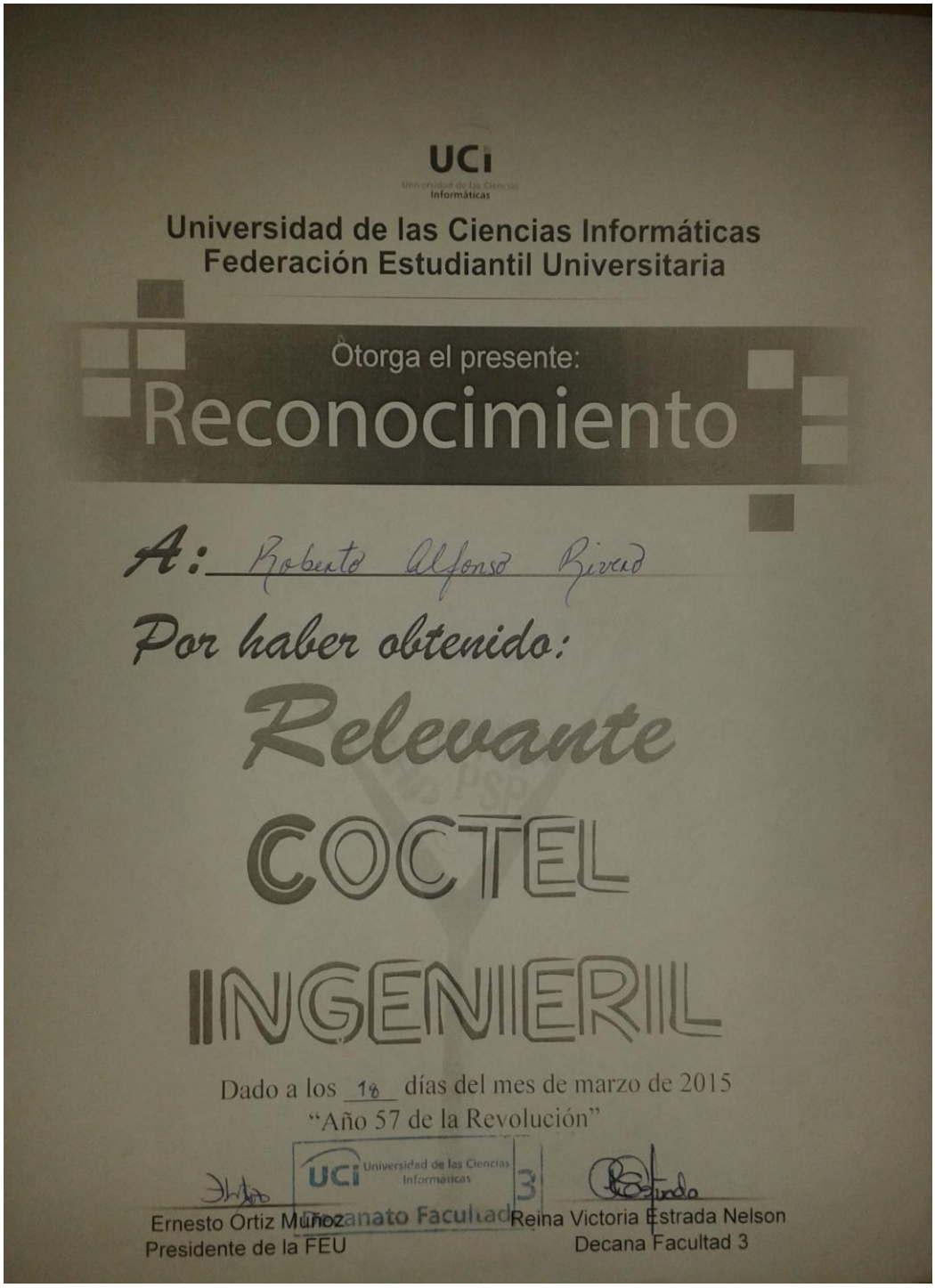


Figura 34: Certificado de la JICI a nivel de facultad (Roberto).

Anexo 8. Certificado de la JICI a nivel UCI.

Figura 35: Certificado de la JICI a nivel UCI.

Anexo 9. Certificado de Jornada Científica Estudiantil Nacional.

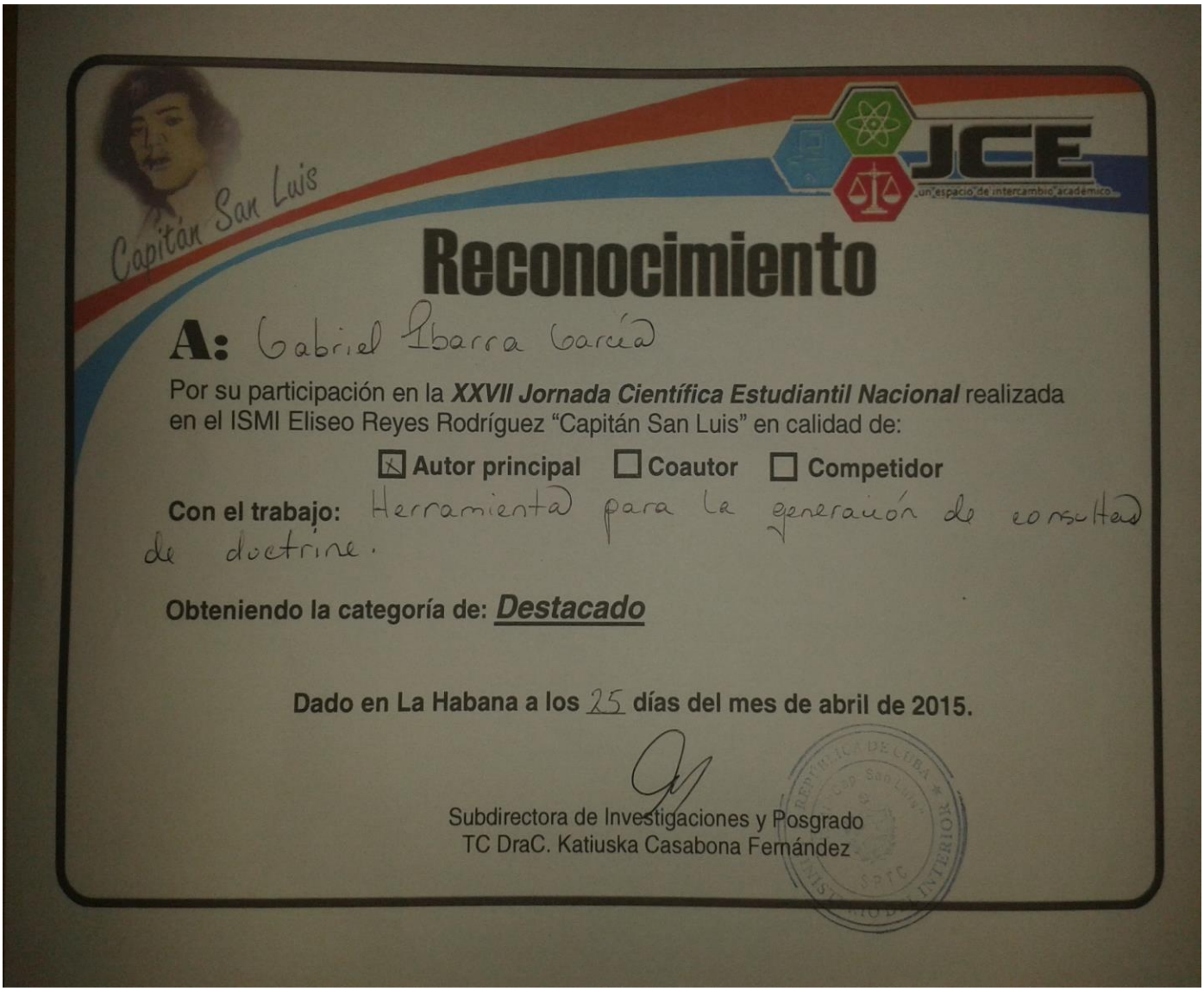


Figura 36: Certificado de Jornada Científica Estudiantil Nacional.

Anexo 10. Certificado de Jornada Científica Estudiantil.

Figura 37: Certificado de Jornada Científica Estudiantil.