Facultad 15

Título: “Desarrollo de una Herramienta generadora de ficheros de mapeo para la persistencia de esquemas de objetos relacionales con los frameworks NHibernate, Hibernate y Propel”.

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

Autor(es): Yelenny Hernández Naranjo

José Yoel Henry Enrique

Tutor(es): Ing. Iosmel Rodríguez Lorenzo

Ing. Alejandro Casanova Mutis

Ciudad de La Habana, junio de 2010

“Año 52 de la Revolución”

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los 26 días del mes de Junio del año 2010.

Yelenny Hernández Naranjo

_____

Firma del Autor

José Yoel Henry Enrique

_____

Firma del Autor

Ing. Iosmel Rodríguez Lorenzo

_____

Firma del Tutor

Ing. Alejandro Casanova Mutis

_____

Firma del Tutor

AGRADECIMIENTOS

Antes que nada agradecer a mi familia que es lo más grande que tengo en esta vida, en especial a mi mamá, mi papá y mi tía Tomasita.

A mi abuela Sofía y mi tío Noel que les hubiera gustado mucho estar en este momento tan especial de mi vida.

Agradecerle a mis Tutores Iosmel y Alejandro que sin ellos no hubiera sido posible este trabajo, en fin a todo aquel que de una forma u otra ha contribuido a la realización del presente trabajo, en especial a Iosmel que me ha ayudado en los cinco años de mi carrera.

Agradecer a todos aquellos que han venido a mi lado en estos cinco años, a mis hermanos Fabián, Argenis, Iván, Elpidio y Piry.

A mis amigos de aula y cuarto, en especial a Omar y Sariol.

A mi novia Aleidy que ha significado mucho para mí su compañía incondicional.

A mi compañera de tesis que no quiero una mejor que ella.

Para terminar a todo aquel que ha echado un granito para formar la persona que soy hoy.

José Yoel

AGRADECIMIENTOS

En mi vida existen dos soles que sin ellos no creo que existiría la persona que les escribe hoy, esas son mis dos reinas, mi mamá Dalia y mi hermanita Yinelis las AMO mis tesoros.

A mi papá que su ejemplo me ha guiado para llegar a ser alguien en esta vida, que tantos tropiezos nos impone.

A mis tíos Ariel, Niurys y Damarys que han sido como mis padres y me han brindado cariño y amor incondicionalmente, los quiero mucho.

A mi abuela Caridad, te quiero mi viejita linda.

A mis hermanos Sergio y Maylín, gracias mis chiquiticos.

A toda mi familia que forma un eslabón importante en el largo camino que he andado y ojalá me acompañen en los momentos buenos y malos que aún me quedan por recorrer, gracias por estar ahí.

A mi novio Abel que ha formado parte importante de mi universidad, brindándome su amor, su confianza y enseñándome que en la vida real se ama sin miedos y todo es posible, te quiero mi vida.

A mis amigas o hermanas de la universidad que aunque no llevan mi sangre significan mucho para mí.

A mis tutores Iosmel y Alejandro que han sido más que tutores, amigos y me han brindado el apoyo más grande del mundo sin ellos esto no hubiese sido posible.

A mi compañero de tesis que me si me dieran a escoger otra vez, lo escogería sin dudas.

A los amigos y amigas que me han acompañado durante estos cinco años les doy gracias a todos por ser parte de mi carrera.

A cada ser especial que compartió conmigo en estos cinco años y contribuyó dando su pedacito le agradezco de todo corazón y con ellos está todo mi cariño.

Yelenny

DEDICATORIA

Quiero dedicarle el presente trabajo de diploma a mi familia que tanto ha confiado en mí y me brinda fuerzas para seguir adelante, en especial a dos personas que han sido la razón de mi existir y que tanto pienso en ellos al dar cualquier paso en mi vida, mi papá que ha confiado en mí incondicionalmente en cualquier circunstancia y a esa mamá que me ha dado el amor más bello y grande de este mundo, para terminar a mi otra mamá que es esa tía Tomasita que ha estado ahí desde que di el primer grito de vida.

José Yoel

Le dedico mi tesis a mi MAMÁ y a mi HERMANA que son las personas más importantes de mi vida y que por ellas estoy buscando un sueño, ser una Ingeniera para que siempre se sientan orgullosas de mí como yo los estoy de ellas, gracias por darme su amor y cariño y acompañarme en las buenas y malas sin esperar nada a cambio. A mi tío Ariel, mis tías Niurys y Damarys, mi abuela y a mi papá les agradezco por formar parte de mi vida.

Yelenny

RESUMEN

Las herramientas generadoras de ficheros son muy usadas en los proyectos productivos en la persistencia de esquemas mediante los frameworks Hibernate, NHibernate y Propel. En los proyectos de Registro y Notarias y Tribunales Populares Cubanos se cuenta con una herramienta llamada N-HiberGen desarrollada por los tutores del presente trabajo. Debido al desarrollo de los sistemas informáticos ha surgido la necesidad de adicionar nuevas funcionalidades a la herramienta existente, con el objetivo de lograr que el tiempo empleado en la implementación de la Capa de Acceso a Datos sea menor.

El presente trabajo realiza un análisis de cómo son generados actualmente los ficheros de mapeo para la persistencia de esquemas usando los frameworks NHibernate, Hibernate y Propel con el objetivo de lograr un producto propio con nuevas funcionalidades buscando mejoras a la herramienta existente.

Luego de un análisis del estado del arte donde se realizó un estudio de las herramientas existentes para la generación de ficheros de mapeo y basado en los inconvenientes que presentan estas herramientas, se realizó el diseño de clases del sistema, dando respuesta al modelo de arquitectura seleccionado, partiendo de lo expuesto en la herramienta anterior.

El principal propósito de este trabajo está enmarcado en desarrollar un sistema que garantice en mayor medida al equipo de desarrollo una eficiente generación de los ficheros necesarios para la persistencia de esquemas mediante los frameworks utilizados.

Índice

Introducción

En la actualidad, para la persistencia de esquemas son usados frameworks, tales como, NHibernate, Hibernate y Propel. Estos son utilizados por los equipos de desarrollo de proyectos como Registros y Notarías para el caso de NHibernate y Propel para el proyecto Tribunales Populares Cubanos, en las aplicaciones en desarrollo que realizan la persistencia de su información en bases de datos relacionales. Para el uso de estos frameworks es necesaria la creación de ficheros de mapeo y sus respectivas entidades persistentes del esquema relacional con el que se necesite interactuar. Existen herramientas que generan estos ficheros de mapeo, pero no cubren todas las necesidades de los equipos de desarrollo debido a que presentan una serie de dificultades en cuanto a las salidas de los ficheros generados, la posibilidad de personalización del mapeo, el mapeo de procedimientos almacenados y la sincronización del esquema relacional y los procedimientos, conllevando esto a que las salidas sean corregidas por los desarrolladores de forma manual; proceso tedioso y que afecta el tiempo estimado de implementación de los sistemas informáticos. En la actualidad, el principal problema que presentan los equipos de desarrollo de los proyectos Registros y Notarías y Tribunales Populares Cubanos, está dado por la no existencia de una herramienta que se corresponda con las necesidades actuales de los desarrolladores, en la generación de ficheros de mapeo para la persistencia de información con el uso de los frameworks NHibernate, Hibernate y Propel.

Situación problemática:

En la Facultad 15 no se cuenta con una herramienta que corresponda con las necesidades de los desarrolladores de los proyectos productivos del Centro de Gobierno Electrónico (CEGEL) en la generación de ficheros de mapeo para la persistencia de objetos relacionales con los frameworks NHibernate, Hibernate y Propel; estas herramientas no permiten la personalización del mapeo para todos los frameworks, el mapeo de procedimientos almacenados, además que no existe una estandarización de las herramientas que brindan soporte para la generación de ficheros para los tres frameworks utilizados, lo que provoca que el trabajo de los desarrolladores sea más tedioso en los diferentes proyectos puesto que las salidas tienen que ser corregidas manualmente en muchos de los casos, independientemente que no se adaptan en su totalidad a las necesidades existentes.

Problema.

Las herramientas generadoras de ficheros de mapeo utilizadas en el centro de gobierno electrónico (CEGEL) no corresponden con las necesidades de los desarrolladores a la hora de personalizar el mapeo lo que está provocando que el trabajo sea más engorroso y un mayor tiempo estimado para la implementación de la Capa de Acceso a Datos.

Objeto de estudio.

Proceso de desarrollo de software.

Campo de acción.

Persistencia de objetos relacionales mediante frameworks.

Objetivo general.

Diseñar e implementar una herramienta capaz de generar los ficheros de mapeo de objetos relacionales para la persistencia de esquemas con los frameworks NHibernate, Hibernate y Propel, partiendo de una ya existente(N-HiberGen), mejorando y completando las funcionalidades requeridas.

Hipótesis.

Si se realiza el diseño e implementación de nuevas funcionalidades a la herramienta (N-HiberGen) para los frameworks de persistencia NHibernate, Hibernate y Propel para los gestores PostgreSQL, Oracle y SQL Server que se correspondan con las necesidades del mapeo y sincronización de los procedimientos almacenados y la sincronización del esquema relacional, se logrará reducir el tiempo estimado en la implementación de la Capa de Acceso.

Objetivos específicos.

- Realizar la fundamentación teórica.
- Seleccionar una metodología de desarrollo para el diseño del sistema atendiendo a la tecnología seleccionada y al problema específico a resolver.

- Diseñar e implementar los algoritmos que permitan el mapeo de procedimientos almacenados, la sincronización de los procedimientos almacenados y del esquema relacional y generación de la solución.
- Validar los resultados obtenidos del sistema desarrollado.

Posibles resultados:

Una herramienta para la generación de ficheros de mapeo, obteniendo de esta forma un producto propio, que permita estandarizar y facilitar este proceso para los desarrolladores de software de los proyectos Registros y Notarías y Tribunales Populares Cubanos.

Capítulo 1. Fundamentación Teórica

1.1. Introducción

En este capítulo se hará referencia al uso de los frameworks para persistencia de Objetos Relacionales buscando resolver la lógica de la persistencia de datos, se explicarán frameworks tales como NHibernate, Hibernate y Propel, características y ventajas de estos, haciéndose referencia además a las herramientas generadoras de código que existen en la actualidad para dar soporte a estos frameworks. Se mencionarán metodologías de desarrollo de las cuales se ha hecho un análisis con el fin de seleccionar una que se adecue a las características del proyecto.

1.2.Framework de Persistencia de Objetos Relacionales.

1.2.1. Framework

Es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables (1)

Genéricamente visto es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación, pero si se precisa detallar que algunos frameworks están regidos por el paradigma MVC (Model-View-Controller) que “separa en la aplicación la gestión de los datos, las operaciones, y la presentación”. Otros frameworks pueden llegar al detalle de definir los nombres de ficheros, su estructura, las convenciones de programación (2).

Los frameworks no tienen por qué estar obligatoriamente enlazados a un lenguaje específico, nada impide que sean ligados a más de un lenguaje.

“Un Framework es una mini-arquitectura reutilizable que provee la estructura genérica y el comportamiento para una familia de abstracciones de software, junto con un contexto formado por metáforas que especifican las colaboraciones y el uso en un dominio dado” (3).

Entre las características más esenciales se encuentran:

- Propone un modelo de colaboración.
- Reutiliza el diseño y el código.
- Incorpora el conocimiento dominio para el que el framework fue diseñado.

Principio de Inversión del Control

Entre sus ventajas se pueden citar:

- Modularidad y reducción de la complejidad. La aplicación está formada por subsistemas especializados en distintos aspectos fundamentales de toda aplicación (persistencia, presentación, manejo de log).
- Fortaleza al cambio. Los módulos pueden ser evolucionados o cambiados conservando la arquitectura global de la aplicación.
- Documentación. La documentación del framework promueve el uso correcto del mismo y disminuye el esfuerzo necesario para el mantenimiento.
- Estructura**.** El desarrollo basado en frameworks establece una estructura sobre la cual las aplicaciones pueden ser construidas, liberando al desarrollador de tomar el 100% de las decisiones de diseño.
- Distribución de funciones. Permite paralelizar el trabajo de desarrollo ya que la solución puede desarrollarse como un conjunto de piezas independientes que encajarán en el framework usado.
- Eficiencia. El desarrollador puede concentrarse en los requerimientos funcionales de la aplicación.
- Aplicaciones ricas. Posibilidad de dar más funcionalidad a los usuarios de la aplicación.

Los frameworks se construyen para dar flexibilidad y generalidad, tratando de cubrir un dominio entero en vez de problemas determinados. Este enfoque produce un Generador de Aplicación más complejo y más extensible (los puntos calientes) que en los sistemas de software tradicionales. Se alcanza la extensibilidad usando la herencia y la encuadernación dinámica, características comunes en lenguajes orientados al objeto tales como C++, Java, C# etc.(4).

Persistencia

La persistencia es la acción de preservar la información de un objeto de forma permanente y que esta a su vez pueda ser recuperada y quede nuevamente utilizable, la información del sistema sobreviva o persista durante el tiempo que sea requerida (5).

En un sistema de persistencia, el programa es virtualmente eterno, si se para y se lanza de nuevo, este mantiene los mismos datos en memoria, sin que se haya almacenado anteriormente.

1.2.2. Framework de Persistencia.

Toda aplicación informática está compuesta por dos partes, el programa que recupera la información de la Base de Datos y la encargada de almacenar la información que el programa necesita para funcionar correctamente, pero entre éstos existe una contradicción debido a que el desarrollo de ellos ha sido completamente independiente, las bases de datos utilizan un modelo relacional(trata con relaciones y conjuntos) mientras que los programas utilizan el modelo orientado a objeto(trata con objetos, atributos y asociaciones) que difiere completamente al modelo relacional.

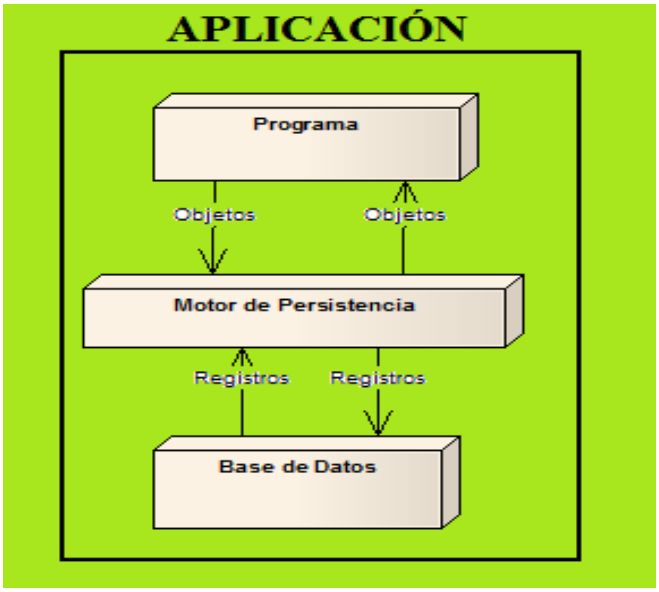


Fig. 1: Motor de Persistencia u ORM

Con el fin de eliminar esta gran brecha o también llamada diferencia objeto-relacional aparecen los motores de persistencias u ORM, que no son más que un traductor entre estos dos tipos de datos distintos. Con un ORM adecuado solo se tendría que definir la forma en que las clases se mapean a tablas, que propiedad se mapea a qué columna y que clase se mapea a qué tabla (6).

La utilización de un framework ofrece:

- Correspondencia de esquemas.
- Identidad de objetos y registros.
- Materialización (transformación de una representación no O.O. en objetos) y desmaterialización.

- Conversor de BD: Responsable de la materialización y desmaterialización.
- Almacenamiento de los objetos materializados en caché.
- Estado de la transacción de objetos para determinar si es necesario almacenar nuevamente los objetos persistentemente.
- Operaciones de transacción: CommityRollback. (7)

1.2.3. Frameworks de persistencia más utilizados

TriActive JDO (TJDO):

- Implementación libre de la especificación JDO 1.0.1 de Sun.
- Implementa el lenguaje de consultas JDOQL, aunque permite consultas SQL directas.
- Auto-creación de todos los elementos necesarios del esquema (tablas, claves, índices).
- Auto-validación de la estructura del esquema en tiempo de ejecución.
- Capacidad de mapear clases de Java a Vistas SQL (8)

IntelliBo:

- Soporte para servidores de aplicaciones: WeLogic, WebSphere, SAP WAS, Oracle, Jboss, Apache Geronimo, Tomcat.
- Soporte para bases de datos JDBC genéricas, y además: Oracle, DB2, MS SQL, informix, HSQLDB, postgreSQL, MySQL.
- Tiene muchas opciones avanzadas para el mapeo de objetos, como división de un objeto simple en múltiples tablas, estrategias de mapeo de herencia (Tabla-Subclase, Tabla-superclase, Tabla-nueva), mapeo serializado, mapeo agregado.
- Opción de verificar las clases y las instrucciones de mapeo.
- Opción para comprobar la consistencia de la estructura de tabla actual.
- Generador automático de código fuente de las clases e instrucciones de mapeo a partir de las estructuras de tablas.
- Generador automático de las estructuras de tablas a partir de las clases e instrucciones de mapeo.
- Integración con Borland Jbuilder, Apache Ant, Eclipse e IBM WebSphere Application Developer.
- Ejemplos de proyectos y documentación comprensible.
- Edición profesional (de pago), edición comunidad (gratuita), plugin para eclipse (8).

FastObjects .NET:

- Motor de persistencia comercial para la plataforma .NET
- Está diseñado para el acceso concurrente a gran escala a las bases de datos, gracias a que ofrece dos estrategias de mapeo (optimista y pesimista).
- Soporte completo para todo el paradigma de la orientación a objetos (interfaces, herencia, polimorfismo y encapsulación de objetos y conjuntos de objetos).
- Posee un lenguaje intermedio que extrae automáticamente todo lo necesario para la creación del esquema a partir del código fuente ya sea en C#, Visual Basic .Net o J#.
- Para las consultas utiliza el lenguaje (estándar-industrial) OQL ObjectQueryLanguaje.
- Soporte completo para todas las características tradicionales de integridad de datos como transacciones, logging y locking.
- Encriptación opcional de los datos e información de la Base de Datos (8).

BEA Kodo:

- Motor de persistencia para Java basado en el código de OpenJPA de Apache (JPA=_ JavaPersistence Api).
- Soporte para la especificación JDO 2.0 y EJB 3.0 JPA.
- Herramienta comercial.
- Es capaz tanto de crear un esquema nuevo a partir de las clases, las clases a partir del esquema o mapear clases existentes en un esquema existente.
- Diseñado para trabajar con o sin aplicaciones servidor.
- Soporte para múltiples bases de datos: Oracle, IBM DB2 y Informix, Microsoft SQL server.
- Integración con Eclipse, BorlandJbuilder, IBM WSAD y NetBeans.
- Soporte para servidores de aplicaciones J2EE: Weblogic, WebSphere, JKBoss (8).

Apache Cayenne:

- Herramienta de código abierto de la fundación Apache.
- Portabilidad entre las bases de datos JDBC.
- Funcionalidad de caché, para hacer el acceso a la Base de Datos más eficiente
- Capacidad de paginación para cargar de la Base de Datos solo los objetos que se necesitan en el momento.

- Tutorial web paso por paso para aprender el manejo de esta herramienta (8).

Oracle top link:

- Motor de persistencia para POJO (Plain Old Java Objects) ejb 2.1, cmp y bmp implementa el API-JPA enfocada a la estandarización de la persistencia objeto-relacional.

- Permite persistencia de objetos Java en Bases de Datos relacionales accesibles utilizando los drivers JDBC.
- También permite la persistencia de objetos Java en Bases de Datos objeto-relacionales como las Bases de Datos de Oracle.
- Soporta Enterprise Information System (EIS); permitiendo la persistencia de objetos Java a fuentes de datos no relacionales que utilizan la arquitectura J2C (J2EE Connector architecture adapter).
- Conversión entre objetos Java y documentos del esquema XML (XSD), usando la arquitectura Java para XML (JAXB).
- Soporte para servidores de aplicaciones IBM Websphere y BEA Weblogic.
- Soporta el caché de objetos para mejorar el rendimiento (8).

Hibernate:

- Herramienta libre bajo licencia LGPL.
- Es una herramienta madura, creada en el 2001, y es una de las más extendidas.
- Es un motor de persistencia para Java, aunque hay una versión para la plataforma .Net llamada NHibernate.
- Compuesto de muchos proyectos como: core (proyecto principal), annotations, entitymanager, shards, validator.
- Tiene un montón de plugins para eclipse y tareas de Ant para ayudar a la utilización y automatización de la persistencia. Están dentro del proyecto Hibernate-tools.
- Hibernate soporta paradigmas de la orientación a objetos como son el polimorfismo y las asociaciones.
- Ofrece un lenguaje de consultas de datos llamado **HQL** (Hibernate Query Language).
- Se pueden crear filtros definidos por el usuario.
- Gracias al proyecto Hibernate-Annotations se pueden utilizar anotaciones de JDK 5.0 junto con los ficheros XML para el mapeo de objetos (8).

NHibernate:

- Modelo de programación natural: NHibernate soporta el lenguaje orientado a objetos; herencia, polimorfismo, composición y el framework de colecciones de .NET incluyendo las colecciones genéricas.
- Hibernate utiliza HQL (Hibernate Query Language) que es un lenguaje de consultas orientado a objetos.
- .NET Nativo: El API de NHibernate usa los nombres y convenciones de .NET.
- Soporte para modelo de objetos de granularidad fina: Una amplia variedad de mapeos de colecciones y objetos que dependen de estas.
- No produce generación de código extra en el procedimiento de construcción.
- NHibernate se ocupa de los dos lados del problema; no solo como llegan los objetos a la Base de Datos sino como extraerlos nuevamente.
- Permite especificar exactamente el SQL que NHibernate usará para persistir los objetos.
- Soporta stored procedures en Microsoft SQL Server.
- Soporta "conversaciones": NHibernate soporta contextos de persistencia y se ocupa de optimizar los lockeos automáticamente.
- Es gratis, open source y está licenciado bajo LGPL (Lesser GNU PublicLicense) (9).

Microsoft Entity Framework:

- Abstracción del Sistema Gestor de Base Datos.
- Brinda Información de mapeo generados por herramientas.
- Utiliza el lenguaje de consultas ESQL.
- Entity Framework incluye un conjunto de herramientas en evolución que generan asignaciones y clases parciales que representan las entidades en el modelo conceptual.
- Entity Framework incluye un proveedor de datos SqlClient actualizado que admite los árboles de comandos canónicos (10).

Propel

- Fácilmente extensible. El empleo de esquemas XML hace posible integrarse fácilmente con otros componentes conducidos por su modelo de datos o ampliar la definición de esquema para incluir atributos para otros componentes.

- Fácilmente customizable. El modelo de objeto generado es construido para ser personalizado. Todas las llamadas internas son hechas para vaciar las clases de trozo que amplían las clases con la lógica generada. Es posible anular el comportamiento en las clases de trozo o proporcionar una lógica propia de encargo de negocio sin sacrificar la capacidad de reconstruir su modelo de objeto en un tiempo posterior.
- Juegos agradables. Estos son un modo de decir que Propel no fuerza a usar sólo sus herramientas. Ejemplo se puede usar los objetos de Criterios para construir preguntas sin escribir SQL, aunque los Criterios no lo hacen todo y también se ha hecho costumbre usar SQL para poblar objetos generados.
- Realmente independiente. El empleo de un proceso para crear SQL personalizado y clases PHP quiere decir que sus interacciones de Base de Datos con Propel son realmente independientes de la Base de Datos.
- Intuitivo, Fiable. La definición de modelo de datos de Propel sigue muy estrechamente la estructura de la Base de Datos subyacente. Esto provee la ventaja de fabricación, Propel brinda una forma fácil de entender y también de comportamiento fiable con modelos de datos muy complejos.
- Avanzado. Propel hace que los problemas relacionales difíciles sean fáciles y eficientes. Por ejemplo, Propel proporciona la capacidad de poblar el objeto relacionado (la llave extranjera) de una sola consulta. Propel provee también apoyo básico de herencia y otros rasgos de OO avanzados.

1.3. Metodologías de desarrollo de Software.

Hoy en día en el desarrollo de un software resulta muy importante la utilización de una metodología de desarrollo, debido a las ventajas que estas nos ofrecen, aunque muchas veces resulte un poco difícil la tarea de seleccionar la más adecuada ya que esta depende en gran medida de las características de cada proyecto a realizar.

El proceso de desarrollo venía asociado al esquema tradicional (roles, actividades y artefactos, incluyendo modelado y documentación detallada) el cual resultaba efectivo para grandes proyectos en cuanto a recursos y tiempo se refería. Con las nuevas necesidades que surgen en el día a día se hace inminente la adaptación y desarrollo de nuevas metodologías ágiles que den respuesta a estas problemáticas manteniendo una alta calidad y reduciendo drásticamente el tiempo de desarrollo. Las metodologías ágiles constituyen una solución a medida para ese

entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto**.**

La creciente comparación entre metodologías tradicionales (también llamadas metodologías pesadas) y las metodologías ágiles es una realidad, debido a que el desarrollo de software está principalmente basado en herramientas que posean una mayor comprensión, se ajusten más a lo que se desea realizar, brinden posibilidades de cambio en caso necesario y todo esto de manera fácil y efectiva, todas estas son características que se encuentran englobadas dentro de las metodologías ágiles.

Dentro de estas metodologías se encuentran:

1.3.1. RUP

Se divide en 4 fases el desarrollo del software:

- **Inicio:** El Objetivo en esta etapa es determinar la visión del proyecto.
- **Elaboración:** En esta etapa el objetivo es determinar la arquitectura óptima.
- **Construcción:** En esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.
- **Transición:** El objetivo es llegar a obtener el release del proyecto.

Cada una de las fases es desarrollada por varias iteraciones según las características del proyecto, los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes, haciendo hincapié en actividades específicas, disciplina bajo las cuales son llevadas las iteraciones en RUP (11):

Disciplina de Desarrollo

- Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Disciplina de Soporte

- Configuración y administración del cambio: Guardar todas las versiones del proyecto.
- Administrando el proyecto: Administrar horarios y recursos.
- Ambiente: Administrar el ambiente de desarrollo.
- Distribución: Hacer todo lo necesario para la salida del proyecto.

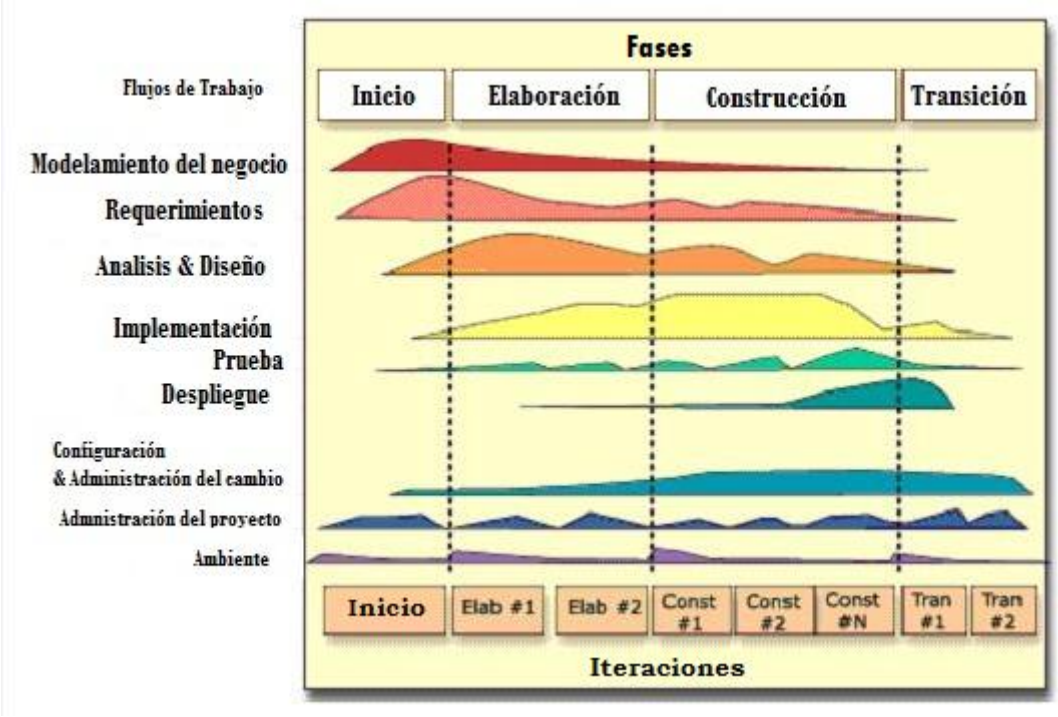


Fig. 2: Fases e Iteraciones de la Metodología RUP

Al terminar cada iteración debe salir un producto entregable por lo que se requiere que se ordenen según su prioridad trayendo esto consigo la retroalimentación en cada iteración.

Los elementos del RUP son:

- **Actividades:** Son los procesos que se llegan a determinar en cada iteración.
- **Trabajadores:** Vienen hacer las personas o entes involucrados en cada proceso.
- **Artefactos:** Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software (11).

1.3.2. Microsoft Solution Framework (MSF)

Es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos (11).

MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Fig. 3: Metodología MSF

MSF tiene las siguientes características:

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre 3 ó 4 personas, así como también, proyectos que requieren 50 personas o más.
- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología (11).

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto:

Modelo de Arquitectura del Proyecto: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones de software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores (11).

1.3.3. XP (ExtremeProgramming)

Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar software (Kent Beck).

En esta metodología los equipos deben ser pequeños y el trabajo se realiza en pareja lo que permite una retroalimentación continua y extensa que garantiza que no se escribirá código sin pruebas de funcionalidad y la “re-fabricación” continua. Es una de las metodologías de desarrollo de software más exitosas en la actualidad, es utilizada para proyectos de corto plazo y equipos pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

Fases de XP.

- Ápice arquitectónico: Conceptualizar de manera general el proyecto. En esta fase suelen presentarse los primeros estimados y es normal que estos sean muy poco precisos.
- Plan de entregas: Se presentan los guiones de usuario y se establecen con más claridad los requerimientos generales del sistema.
- Iteración: En esta fase las iteraciones en XP deben ser notablemente cortas.
- Pruebas de Aceptación: Se comparan los resultados actuales con lo que se esperaba del sistema aunque a medida que se va construyendo el producto se van haciendo entregas continuas que permiten ir dando evaluaciones parciales. (12)

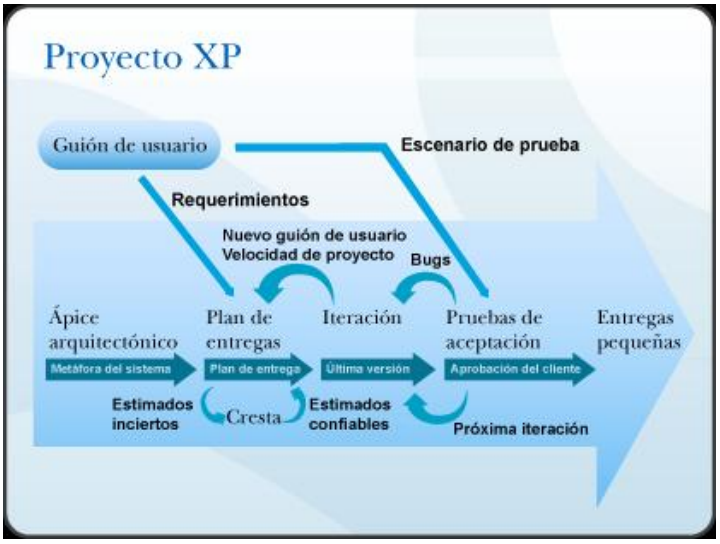


Fig 4. Proceso de XP

Características de XP, la metodología se basa en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que en el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como adelantarse a obtener los posibles errores.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento (11).

Lo que propone XP:

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo (11).

Derechos del Cliente

- Decidir qué se implementa.
- Saber el estado real y el progreso del proyecto.
- Añadir, cambiar o quitar requerimientos en cualquier momento.
- Obtener lo máximo de cada semana de trabajo.
- Obtener un sistema funcionando cada 3 ó 4 meses

Derechos del Desarrollador

- Decidir cómo se implementan los procesos.
- Crear el sistema con la mejor calidad posible.
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos.
- Estimar el esfuerzo para implementar el sistema.
- Cambiar los requerimientos en base a nuevos descubrimientos

Lo fundamental en esta metodología es:

- La comunicación entre los usuarios y los desarrolladores.
- La simplicidad al desarrollar y codificar los módulos del sistema (11).

1.3.4. SCRUM

Scrum define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto a los procesos definidos (cascada, espiral, prototipos, etc.), define un marco para la gestión de proyectos, se centra en priorizar el trabajo en función del valor que tenga para el negocio, maximizando el valor de lo que se construye y acelerando el retorno de la inversión, se emplea en entornos que trabajan con requisitos inestables y que requieren agilidad y flexibilidad, permite incorporar cambios con rapidez y en cualquier fase del proyecto.

Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes o poco definidos, la innovación, la competitividad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto (13)

Está compuesta por dos características fundamentales:

- Se realiza mediante iteraciones llamadas Sprint, cada iteración tiene que proporcionar un resultado completo.
- Establece la realización de reuniones a lo largo de todo el proyecto para coordinación e integración del equipo de desarrollo, además de reuniones diarias.

Roles dentro de Scrum:

- El dueño del producto, que representa la voz del cliente y aporta la visión de negocio. Él se encarga de escribir las historias de usuario, les da prioridad y las ubica en la lista de requisitos del producto.
- El ScrumMaster o facilitador, que tiene como principal papel dejar el camino libre de obstáculos e impedimentos para que el resto del equipo consiga el objetivo del Sprint.
- El equipo, que tiene la responsabilidad de entregar el producto.
- Los usuarios del producto o aplicación.
- Los clientes y vendedores.
- Los gestores y directivos (14).

1.4. Generadores de Ficheros de Mapeo de Objetos Relacionales

1.4.1. NHiberGen

Actualmente en el Proyecto Registros y Notarias es utilizada una herramienta para la generación de ficheros de mapeo llamada NHiberGen, la cual permite: el mapeo de los objetos relacionales,

la configuración de la conexión con los gestores SQLServer y Oracle, la personalización de los objetos, además el mapeo en el esquema (XML) de la clase generada para cada objeto y la configuración de las relaciones de los objetos. Con el transcurso del tiempo y el avance de los sistemas informáticos se hace necesario agregarle nuevas funcionalidades a esta herramienta, las cuales no estaban contempladas en su primera versión.

Nuevas funcionalidades:

- Conexión a servidores PostgreSQL.
- Mapeo de ficheros de mapeo para procedimientos almacenados.
- Sincronización del esquema relacional.
- Sincronización de procedimientos almacenados.
- Generación de ficheros de mapeo para Hibernate y Propel.
- Generación de una solución compilada.

1.4.2. Herramientas de Mapeo de Objetos Relacionales

MyGenerator

Es un generador de códigos open source que dispone de varias plantillas para generar las clases de acceso a datos y los archivos de configuración de NHibernate a partir de la estructura de nuestra Base de Datos (15)

Es una herramienta de desarrollo extremadamente flexible escrito en Microsoft. NET.

Características:

Soporta una variedad de sistemas de DBMS:

- Microsoft SQLServer.
- Oracle.
- Microsoft Access.
- MySQL.
- PostgreSQL.
- Firebird.
- Interbase.

- SQLite.
- VistaDB.

Ventajas

- Excelente para la generación de arquitecturas de ORM para los archivos de asignación Gentle.Net, Opf3, NHibernate, y otros.
- Plantilla idiomas soportados incluyen C #, VB.NET, JScript y VBScript.
- Las plantillas incluyen archivos de apoyo para su reutilización. Posibilidad de paso a través de todas las plantillas sin importar el idioma.
- Capacidad única para proporcionar una interfaz de usuario personalizada en sus plantillas (incluido el pleno apoyo System.Windows.Forms).
- Los archivos de proyecto permitirán regenerar el proyecto en cuestión de segundos.
- Interfaz de línea de comandos proporcionada (ZuesCmd.Exe).
- Meta MyMeta potente API de datos sirve a su Base de Datos de meta-datos.
- Editor de la sintaxis de gran alcance con la ayuda completa de Unicode.
- Plug-In de apoyo que le permiten ampliar la funcionalidad de todas las plantillas. Ventanas plegables.
- Dinámica y asignaciones reemplazables de tipos de bases de datos nativos de la lengua y los tipos de proveedor de datos.
- Soporta los metadatos definidos por el usuario.
- Permite el suavizado de tablas, nombres de columnas, parámetros y más.
- Sintaxis región Perserve le permite preservar los segmentos de código en la regeneración.
- Gentle.NET, Opf3, NHibernate plantillas están disponibles.
- Biblioteca de plantillas en línea para el intercambio y la colaboración (16).

GenWise Studio

Studio GenWise es una plantilla basada en generador de código para. NET 2.0

Genera un período de tres niveles de aplicación ASP.NET Web:

- Capa de persistencia de objetos utilizando NHibernate.
- Business ObjectLayer.

- Interfaz de usuario de capa usando Asp.NET 2.0 Webforms.

Independiente de los productos: Visual Studio 2005 no es necesario. Contiene IDE incluidos.

Posibilidad de ampliación máxima: se puede agregar código personalizado en todas partes entre el código generado, sin perder en el ciclo de la próxima generación.

Máxima productividad: utiliza su actual modelo de Base de Datos como la base para la generación.

Soporte para MS SQL, Oracle, DB2, Postgress, MySql, y iAnywhere de Sybase ASE (17). Los archivos de mapeo de objetos de NHibernate, clases C# y fábricas de objetos de negocio se crean sobre la base de un esquema de Base de Datos.

Excelente persistencia y capa de objetos de negocio para proyectos ASP.NET.

Funcionamiento a la perfección con claves compuestas, tipos nulos y relaciones de mucho a mucho y uno a uno.

Componentes creados por GenWise Studio.

Objetos de negocio o clases de negocio.

Genera propiedades fuertemente tipadas (utilizando tipos de .NET), basado en el repositorio de datos de la tabla, las columnas.

Genera propiedades para todo tipo de relaciones: muchos a uno, muchos a muchos y de uno a muchos.

Mapeo de los archivos XML de NHibernate.

Los archivos XML se generan automáticamente en función de los metadatos almacenados en el repositorio de la Base de Datos.

Fábrica de clases.

Estas fábricas contienen los métodos de acceso a la Base de Datos (Crear, actualizar y eliminar) (18).

ObjectMapper .NET

Es un proyecto de código abierto para la persistencia de objetos. Objeto mapeador relacional con el objetivo de reducir la brecha entre programación orientada a objetos y modelos de Bases de Datos relacionales. Ofrece el estado de la funcionalidad de objeto de arte de mapeo.

Brinda el máximo rendimiento y facilidad de uso, dándole a los desarrolladores la capacidad de construir componentes de negocio con productividad sin precedente y óptimo rendimiento en funcionamiento.

El ObjectMapper.NET utilizará la información de metadatos de modelo para crear un archivo DDL al crear la Base de Datos incluyendo todas las restricciones.

No hay necesidad de mantener un DDL separado y de sincronizar manualmente las propiedades y columnas de la tabla Base de Datos. Todos los contenidos en una sola fuente, eliminando así los obstáculos de mantenimiento y toda una clase de condiciones de error (19).

1.5. Conclusiones

En este capítulo se realizó un estudio de los principales frameworks de persistencia utilizados en el proceso de mapeo de objetos relacionales, se abordó acerca de las herramientas que existen para la generación de ficheros de mapeo, enfatizando el estudio en el análisis de la herramienta utilizada actualmente por el proyecto de Registros y Notarias con el objetivo de agregarle nuevas funcionalidades que cubran las necesidades existentes en los equipos de desarrollo, buscando agilizar el trabajo para los desarrolladores de software. Debido a que es una herramienta y partiendo de las nuevas necesidades surgidas se llegó a la conclusión de:

- Se continuará trabajando con la plataforma Microsoft.Net y como lenguaje de programación C#, debido a que se le dará continuidad a una herramienta ya desarrollada sobre esas tecnologías.
- El surgimiento de una nueva herramienta lleva a que se plantee una revisión y completamiento del diseño e implementación de la existente, la cual abarque las principales ventajas de la creada y cubra la mayor parte de las necesidades de los desarrolladores de software que utilizan dichos frameworks de persistencia, teniendo como puntos positivos la agilización del desarrollo de aplicaciones, así como la estandarización de una herramienta para la generación de ficheros de mapeos propia de

la facultad 15. De esta forma, se logrará una estandarización a la hora de trabajar en los proyectos productivos cuando se realicen las tareas de mapeo de objetos relacionales, independientemente del framework, plataforma o lenguaje que se utilice.

Después de haber hecho un estudio detallado sobre algunas metodologías de desarrollo y teniendo en cuenta las características del proyecto y el tamaño del equipo de desarrollo se decide el uso de la metodología de desarrollo ágil XP (ExtremeProgramming) debido a**:**

- XP permite la retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.
- XP utiliza como una de sus prácticas fundamentales la programación en pareja lo que facilita que el código sea revisado constantemente evitando una mayor cantidad de errores y dando como resultado un código eficaz y de alta calidad.

Capitulo 2: Descripción de la Solución Propuesta

2.1 Introducción

En este capítulo se realizará la descripción de la solución, se exponen características de la arquitectura, la vista vertical de ésta, así como los patrones que fueron utilizados en cada una de las capas. Se explicarán aspectos importantes como son las características de los catálogos de datos, las interfaces de usuario y las herramientas utilizadas en el desarrollo. Se realizará todo el proceso de desarrollo de la metodología seleccionada en el capítulo anterior como respaldo al software desarrollado, en este caso ExtremeProgramming (XP) documentándose cada una de sus fases.

2.2 Descripción de la solución

La solución es una aplicación de código abierto enfocada generación de ficheros de mapeo para la persistencia de objetos relacionales, centrada en la persistencia de objetos mediante los frameworks NHibernate, Hibernate y Propel, que ofrece funcionalidades de mapeo que lo sitúan por delante de los mapeadores de objetos relacionales utilizados por los equipos de desarrollo del centro.

La herramienta se diseñó con el principio de brindar al desarrollador de aplicaciones que necesiten persistir su información con alguno de los frameworks antes mencionados, un ambiente de desarrollo fácil de usar y amigable a la vista del desarrollador. La herramienta proporciona la posibilidad de construir los componentes de negocio según la lógica implementada en el sistema que se esté desarrollando y con un buen rendimiento en el funcionamiento.

Puntos a tener en cuenta:

- Objetivo: persigue como objetivo fundamental cubrir las necesidades que tienen los equipos de desarrollo del centro, que utilizan los frameworks NHibernate, Hibernate o Propel para la persistencia de datos al generar los ficheros de mapeo de los esquemas relacionales con las herramientas que cuentan para ello, brindándole la posibilidad al desarrollador de configurar a su gusto la forma en que el sistema generará las salidas para los ficheros de mapeo. Para complementar lo antes enunciado el sistema ofrece la posibilidad al desarrollador de personalizar su propio mapeo, en cuanto a relaciones que existen entre los esquemas presentes en las Bases de Datos, relaciones como

generalización/especialización, uno a uno y uno a muchos; generar ficheros de mapeo para procedimientos almacenados para el framework NHibernate con el gestor Oracle y la sincronización de los procedimientos almacenados y el esquema relacional, además de la generación de la solución, respondiendo o cubriendo con estas funcionalidades las principales necesidades que presentan los equipos de desarrollo.

- Alcance: teniendo en cuenta los principales gestores de Bases de Datos utilizados en el desarrollo de aplicaciones informáticas en el centro, el sistema permite conectividad para SQL Server, Oracle y PostgreSQL, permitiendo el mapeo de los esquemas creados en dichos gestores y el mapeo de procedimientos en Bases de Datos creadas en el gestor Oracle. Además, atendiendo a los principales lenguajes utilizados para el desarrollo de estos sistemas, las salidas se generan para lenguajes C# en sus diferentes versiones (2003-2005), para Java y PHP.
- Funcionamiento: para esta versión de la herramienta, el mapeo de los objetos relacionales lo realiza partiendo de un esquema relacional en una Base de Datos, brinda como inicio del proyecto una interfaz en la que puede configurar la conexión con uno de los gestores antes mencionados y obteniendo seguido de esto la lista de relaciones del esquema relacional, de los que seleccionará a los que se le necesite realizar el mapeo, puede ser almacenado y editado. Cuenta con una serie de interfaces de usuario para la personalización de los objetos, en las que podrá editar a gusto las salidas de las clases generadas para dichos objetos. Para cada objeto el sistema genera una clase persistente (*.cs, *.java, *.php), la interfaz de la clase con el prefijo “I” (anexo # 1) seguido del nombre de la clase que es el nombre del objeto, además el mapeo del esquema correspondiente a la clase en archivos XML o YML según el framework para el cual se genera el mapeo. Cuenta además con interfaces para la configuración de relaciones de los objetos, en las que se muestran las posibles relaciones que puede tener el objeto con otros objetos mapeados, divididas en tres, las relaciones de herencia o generalización/especialización, de uno a uno y de muchos a muchos respectivamente, las que el desarrollador podrá configurar a su gusto para el objeto al que se le esté realizando la personalización del mapeo atendiendo a la lógica de negocio implementada en el sistema. Incorpora flujos para el mapeo de procedimientos almacenados y sincronización del esquema relacional y los procedimientos. El mapeo y sincronización de procedimientos se realiza con Bases de Datos en gestores Oracle y para el framework NHibernate, mientras que la sincronización de esquemas relacionales se realiza para todos los gestores contemplados en la herramienta. Como salida general, el sistema entrega una solución compilada con el

framework.Net versión 2.0 que contendrá una Dynamic Link Library (DLL)que agrupará paquetes o namespaces contenidos por clases persistentes y sus correspondientes interfaces, el mapeo de estas clases persistentes, los Data Objects Access de los procedimientos almacenados y el mapeo de estos procedimientos.

- Vistas Previas: el sistema brinda al usuario dos vistas previas, la primera vista es la del código generado de la clase, para los lenguajes mencionados anteriormente y el esquema (XML o YML) generado para la clase y la segunda vista es un diagrama de clases de la jerarquía de clases de un objeto, en el que se incluyen las relaciones mapeadas para dicho objeto.

2.3 Descripción de la Línea Base da la Arquitectura

La arquitectura de software de un programa es la estructura o estructuras del sistema, la cual comprende los componentes de software, las propiedades externas visibles de estos elementos y las relaciones entre ellos. El objetivo de este epígrafe es definir una arquitectura flexible y consistente, basándose en la utilización de patrones de diseño para la construcción de las aplicaciones. Partiendo del popular modelo de aplicación de 3 capas, se pretende alcanzar un modelo más refinado que evolucione, describiendo ciertas recomendaciones y pautas de diseño (18).

2.3.1 Separación lógica en capas

Al plantear el diseño de esta aplicación, el primer paso es conseguir separar conceptualmente las tareas que el sistema debe desempeñar entre las distintas capas lógicas y en base a la naturaleza de tales tareas. Se ha de partir de la separación inicial en tres capas, diferenciando que proceso de los que hay que modelar responde a tareas de presentación, cual a negocio y cual a acceso a datos. En caso de identificar algún proceso lógico que abarque responsabilidades adjudicadas a dos o más capas distintas, es probable que dicho proceso deba ser dividido en subprocesos iterativamente, hasta alcanzar el punto en el que no exista ninguno que abarque más de una capa lógica.

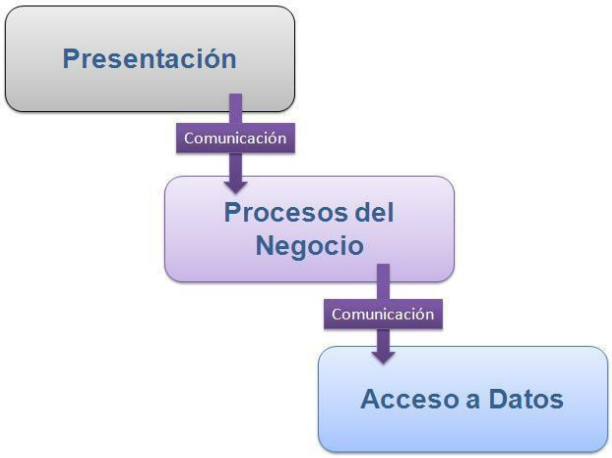


Fig. 5: Modelo de aplicación en 3 capas.

2.3.2 Capa de presentación

Como se dijo anteriormente, es la responsable de todos los aspectos relacionados con la interfaz de usuario de la aplicación. Así, en esta capa se resuelven cuestiones como:

- Navegabilidad del sistema: mapa de navegación.

- Formateo de los datos de salida: resolución del formato más adecuado para la presentación de resultados.

- Validación de los datos de entrada: en cuanto a formatos, longitudes máximas.

- Interfaz gráfica de usuario (18).

2.3.3 Capa de negocio

En esta capa es donde se deben implementar todas aquellas reglas obtenidas a partir del análisis funcional del proyecto. Así mismo, debe ser completamente independiente de cualquiera de los aspectos relacionados con la presentación de la misma. Por otro lado, la capa de negocio ha de ser también completamente independiente de los mecanismos de persistencia empleados en la capa de acceso a datos. Cuando la capa de negocio requiera recuperar o persistir entidades o cualquier conjunto de información, lo hará siempre apoyándose en los servicios que ofrezca la capa de acceso a datos para ello. De esta forma, la sustitución del motor de persistencia no afecta lo más mínimo a esta parte del sistema. Deberían poder agregarse

nuevos gestores de Bases de Datos por un conjunto de ficheros de texto sin necesitar tomar ni una línea de código de negocio. Las responsabilidades que conviene abordar en esta capa son:

- Implementación de los procesos de negocio identificados en el análisis del proyecto.
- Control de acceso a los servicios de negocio.
- Publicación de servicios de negocio.
- Invocación a la capa de persistencia.

Los procesos de negocio son los que determinan qué, cómo y cuándo se debe persistir en el repositorio de información. Los servicios ofertados por la interfaz de la capa de acceso a datos son invocados desde la capa de negocio en base a los requerimientos de los procesos en ella implementados.

2.3.4 Capa de acceso a datos

La capa de acceso a datos es la responsable de la gestión de la persistencia de la información manejada en las capas superiores. En el sistema más específicamente es la responsable de cargar toda la información necesaria para el posterior procesamiento de estos datos por la capa de negocio. La misma de igual forma debe estar preparada por su diseño, para aceptar la interacción con nuevos gestores de Base de Datos con la menor cantidad de cambios o reestructuraciones posibles (18).

2.3.5 Comunicación entre capas. Desacoplamiento.

La evolución del modelo 3-capas al modelo n-capas pasa por la incorporación de las capas intermedias que permiten desacoplar las primitivas y distribuirlas. El modelo recomendado y uno de los más utilizados en el mundo actual es el que se obtiene implementando el patrón de diseño Fachada propuesto por Gang of Four (GOF), muestra a la capa superior el conjunto de servicios que ofrece la capa inferior, al mismo tiempo que posibilita la invocación remota de los mismos de forma transparente al usuario del servicio. De esta forma, la separación lógica entre capas pasa a ser también física, y se consigue dotar a la aplicación de escalabilidad vertical. No obstante, este tipo de arquitectura presenta un problema importante. Cuando la aplicación no está distribuida verticalmente, porque quizá no sea necesario aún, y se encuentra desplegada en una sola máquina, se están realizando invocaciones Java Remote Method Invocation (RMI) al localhost por cada solicitud de servicio que se realice de una capa a otra. Esto, lógicamente, presenta un coste innecesario que puede provocar un descenso de rendimiento importante en el sistema, pero como en el caso específico de la solución propuesta, por sus características de ser

una herramienta centralizada siempre en una misma máquina, no tiene porque aparecer dicho problema en la solución.

2.3.6 Aspectos a tener en cuenta

Los modelos de la arquitectura deben responder a una serie de aspectos para su posible validación ya sea por grupos especializados de calidad o para la satisfacción del grupo de desarrollo, que la arquitectura a implementar asegure la fiabilidad, usabilidad y posterior éxito del software a desarrollarse, así como el aseguramiento al cliente de la salida del producto acordado respondiendo a los intereses por los cuales el mismo decidió adoptar dicha solución informática

Por ejemplo mencionaremos algunos de los aspectos que se van a tener en cuenta:

- Descripción de atributos de calidad observables vía ejecución.
- Descripción de atributos de calidad no observables vía ejecución.

Ahora cada sistema debe adoptar cuales aspectos deben responder como prioridad, es decir, en cuales aspectos debe responder con más fuerza el sistema que se desea desarrollar, atendiendo esta al tipo de software que se esté desarrollando y a las características específicas de dicho software. Debido a las características de la aplicación que se está desarrollando, se adopta como principales aspectos a tener en cuenta (18):

- Funcionalidad
- Integrabilidad
- Modificabilidad
- Mantenibilidad
- Reusabilidad

2.4 Vista vertical de la propuesta de arquitectura

La vista vertical de la arquitectura del sistema que se propone, es un estilo en capas, estructurado de la siguiente forma (figura 6); una primera capa, la de acceso a datos, que se encargará de la comunicación con los distintos servidores de datos que soporte la aplicación, esta capa está diseñada de forma tal que la incorporación de un nuevo gestor de Base de Datos solo lleve como cambio en su estructura de paquetes, la adición de un conjunto de clases cuyo propósito es la interacción con el nuevo gestor. Las clases que forman esta capa implementarán un conjunto de funcionalidades presentes en el “Servicio de Datos” basándose para esto en el

patrón Abstract Factory / Fábrica Abstracta (anexo 2) que provee la aplicación de la abstracción en el momento de crear familias de objetos para la interacción con un gestor de Base de Datos en específico. Siendo el creador de las familias de objetos una interfaz común, la cual será la proveedora de la interacción a capas superiores con la capa de acceso a datos (20). Las funcionalidades que se implementan en esta capa garantizarán un estándar de comunicación de las distintas clases de la capa de acceso a datos con sus respectivos gestores de datos a interactuar así como simplificar casi a la máxima expresión los posibles cambios que puedan traer en la capa de negocio la inclusión de un nuevo gestor de Base de Datos. Las funcionalidades presentes en el "Servicio de Datos" a su vez sirven de comunicación entre la capa de negocio y la capa de acceso a datos, para lo que se implementa el patrón Facade / Fachada (anexo 3) el cual se basa en proporcionar al programador una clase sencilla con un grupo de métodos, uno para cada operación permitida y de modo que sean estos métodos los que internamente hagan las operaciones con el fin de llevar a cabo la correcta lógica de la aplicación (21). La capa de negocio contiene las entidades del modelo de diseño que dan descripción y modelado al negocio presente en la aplicación que se propone, así como las entidades de tipo gestoras que mantienen todo el control y seguimiento de los distintos hilos de negocios que se crean. En esta capa se utiliza el patrón Singlenton / Solitario (anexo 4) para la generación de una instancia única de los gestores de negocio a utilizar (22).

También se cuenta con la capa de "Presentación", que contiene todos los componentes y diversos controles elaborados para el desarrollo del software, así como las distintas interfaces de usuarios y pautas de diseños aplicadas a las mismas, están presentes también en dicha capa los distintos flujos de navegación que seguirá el usuario mediante la interacción con la aplicación. Otra capa presente en la solución es la de "Generación de Salidas", en la misma está presente una serie de componentes y entidades encargadas de generar los ficheros de salidas. Similar a la capa de acceso a datos, en esta las entidades presentes y encargadas de generar ficheros específicos deben implementar una serie de funcionalidades presentes en el "Servicio de Generación", que el mismo contiene una serie de funcionalidades que garantizan estándares de implementación por parte de las entidades de la capa de generación de salidas, además garantizan reducir a la mínima expresión los posibles cambios que puedan originar la inclusión de un nuevo tipo de fichero a generar, por ejemplo un nuevo lenguaje que se quiera incorporar. Verticalmente a las capas antes descritas se encuentra el tratamiento de errores, presente por un módulo a utilizar en todo el sistema, el mismo garantiza de forma dinámica la incorporación de nuevos tipos de excepciones, así como mantener un log de todas las excepciones generadas por el sistema con los datos precisos para el posterior análisis por parte del equipo de desarrollo.

También de forma vertical a la arquitectura propuesta se encuentra el .Net framework, como framework a utilizar por todo el sistema.

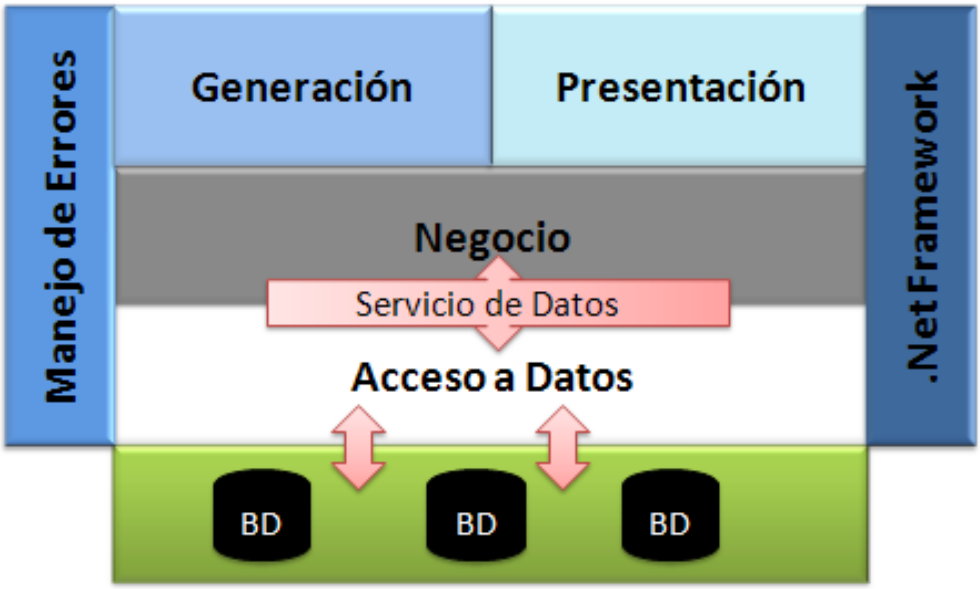


Fig.6: Vista vertical de la Arquitectura

2.5 Herramienta para el desarrollo.

2.5.1 Visual Studio 2005

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones Web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones Web ASP y Servicios Web XML.

C# es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework, es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas innovaciones, permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Visual Studio presenta un editor de código con grandes prestaciones para C#, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa (23).

2.5.2 Framework. NET

El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican enormemente el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución altamente distribuido, que permite crear aplicaciones robustas y escalables (24).

Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (Common Language Runtime)

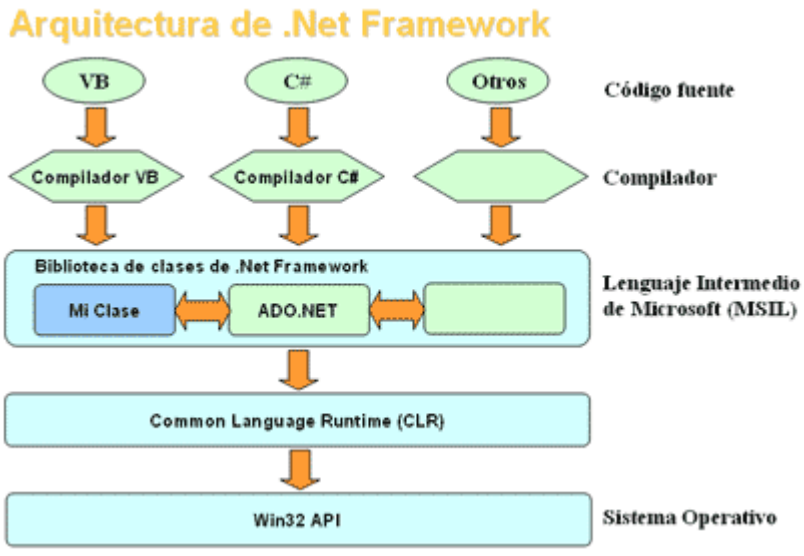


Fig. 7: Arquitectura de .Net Framework

.Net Framework soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Existen más de 30 lenguajes adaptados a .Net, desde los más conocidos como

C# (C Sharp), Visual Basic o C++ hasta otros lenguajes menos conocidos como Perl o Cobol (24).

El CLR es el motor de ejecución de las aplicaciones .NET, lo que en Java sería la máquina virtual, este motor se encarga de ejecutar todo el código .NET para ello ha de ser en dicho lenguaje. El CLR es el encargado de convertir este lenguaje intermedio en leguaje máquina del procesador, esto normalmente se hace en tiempo real por un compilador Just-In-Time (JIT) que lleva incorporado el CLR.

Biblioteca de clases del .NET Framework es la piedra angular de cualquier desarrollador de .NET, es un rico conjunto de clases, interfaces, tipos que simplifican y optimizan el desarrollo de aplicaciones .NET además de proporcionar acceso a la funcionalidad del sistema. Como desarrolladores, el dominio de este conjunto de clases es vital para un buen desarrollo en .NET.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad lo que posibilita que toda la biblioteca .Net esté centralizada bajo el mismo espacio de nombre (System) (24)

La biblioteca de clases de .Net Framework incluye, entre otros, tres componentes clave:

- ASP.NET para construir aplicaciones y servicios Web.
- Windows Forms para desarrollar interfaces de usuario.
- ADO.NET para conectar las aplicaciones a bases de datos.

ASP.NET es la parte del .NET Framework dedicada al desarrollo web. A través del servidor web (IIS) las aplicaciones ASP.NET se ejecutarán bajo el CLR y podrán usar el conjunto de clases del .NET Framework para desarrollarlas, obteniendo así una versatilidad y una potencia nunca antes conseguida en las aplicaciones ASP (25).

Ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Brindan la posibilidad de resolver problemas de aplicaciones actuales que en muchos casos tienen que tratar con diferentes archivos binarios (DLL´s), elementos de registro, conectividad abierta a bases de datos (ODBC), etc.

2.6 Metodología XP

La metodología XP posee algunas similitudes con el Proceso Unificado de Desarrollo (RUP) y en algún modo puede ser considerado una versión abreviada del RUP con modificaciones, sin embargo, XP no utiliza representaciones gráficas (diagramas) para el análisis y diseño de los procesos. Basado en los métodos que utiliza XP, se espera que el código producido sea de alta calidad. El ciclo de vida ideal de XP consiste de seis fases que serán abordadas en el capítulo, ellas son: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (26).

2.6.1 Fase: Exploración

En esta fase, los clientes plantean a grandes rasgos las Historias de Usuario (HU) que son de interés para la primera entrega del producto. Al mismo tiempo, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (27).

2.6.1.1 Historias de Usuario

Las Historias de Usuario (HU) son la técnica utilizada en XP, representan una breve descripción del comportamiento del sistema, emplea terminología del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema, se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, son la base para las pruebas de unidad y presiden la creación de las pruebas de aceptación.

Se trata de tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Las HU son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración. Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto (27). Atendiendo a las necesidades del proyecto y basado en un estudio previo se define la siguiente estructura para las HU siendo la casilla de observación opcional, solo se utilizará en casos específicos.

Historia de Usuario	
Nombre:	
Número:	**Usuario:**
Dependiente:	
Prioridad:	**Puntos Estimados:**
Descripción:	
Observaciones:	

Tabla # 1: Estructura de una HU

Donde:

- Nombre: Nombre de la Historia de Usuario
- Número: Identificador de la HU.
- Usuario: Creador de la HU.
- Puntos Estimados: Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a cuatro días en la programación, por lo que los puntos estimados varían en dependencia de la complejidad de la HU.
- Prioridad: La prioridad refiere a la importancia de la Historia de Usuario, medida en una escala Baja, Media o Alta.
- Dependiente: Cada HU debe ser independiente pero a veces esto no se logra, en esta casilla se detalla esta característica.
- Descripción: Descripción sintetizada de la HU.
- Observaciones: Información de interés en caso de que sea necesaria.

Durante el desarrollo de esta fase se identificaron varias HU dentro de las cuales se detallan las de principal importancia:

Historia de Usuario
Nombre Historia de Usuario: Conexión a bases de datos y extracción de esquemas

relacionales en servidores PostgreSQL.	
Número: 1	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 2
Descripción: Se solicitan los datos de configuración de conexión con un servidor en PostgreSQL, se establece la conexión, se extrae la información de las bases de datos creadas en el servidor y para cada una de estas el esquema relacional modelado en ella.	

HU 1: Servidor PostgreSQL y extracción de información.

Historia de Usuario	
Nombre Historia de Usuario: Conversión del esquema relacional a la estructura de negocio que soporta esta información en la aplicación.	
Número: 2	**Usuario:** Desarrollador
Dependiente: 1	
Prioridad: Alta	**Puntos Estimados:** 0.5
Descripción: Luego de la extracción del esquema relacional de una Base de Datos en PostgreSQL, se convierte esta estructura a la estructura de entidades de negocio que soportará la información del esquema relacional en la aplicación.	

HU 2: Conversión del esquema relacional a la estructura del negocio de la aplicación.

Historia de Usuario	
Nombre Historia de Usuario: Generación de los esquemas de mapeo en fichero *.YML para el ORM Propel.	
Número: 3	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 1
Descripción: Partiendo de la personalización previa del mapeo del esquema relacional extraído de un gestor Oracle, SQL Server o PostgreSQL, se confecciona el mapeo en el fichero *.YML	

para cada uno de los objetos mapeados.

HU 3: Generación de esquemas de mapeo fichero *.YML para el ORM Propel.

Historia de Usuario	
Nombre Historia de Usuario: Mapeo de procedimientos almacenados para el Gestor Oracle.	
Número: 4	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 7
Descripción: Se extraen procedimientos almacenados de un SGBD Oracle, luego de extraídos los procedimientos, se seleccionan los que se vayan a mapear y de estos se extrae la información necesaria para el mapeo (parámetros, dirección de los parámetros, tipos de datos, etc.).	

HU 4 Mapeo de procedimientos almacenados para el Gestor Oracle.

Historia de Usuario	
Nombre Historia de Usuario: Sincronización de Procedimientos Almacenados.	
Número: 5	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 3
Descripción: Se extraen los nuevos procedimientos almacenados, su información, se establece una comparación con los procedimientos almacenados ya mapeados y se realiza la sincronización.	

HU 5 Sincronización de Procedimientos Almacenados.

Historia de Usuario	
Nombre Historia de Usuario: Sincronización del esquema relacional.	
Número: 6	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 3

Descripción: Se extrae el esquema relacional y se compara con el esquema relacional ya obtenido de la Base de Datos anterior y se realiza la sincronización.

HU 6 Sincronización del esquema relacional.

2.6.2 Fase: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, los programadores realizan una estimación del esfuerzo necesario para implementar cada una de ellas. Esto se expresa utilizando como medida el punto, el cual se consideró como 4 días de trabajo en programación real. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las HU que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias, además permiten determinar si una iteración está sobrecargada. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las HU seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación (26) (27).

2.6.2.1 Estimación de esfuerzo por Historias de Usuario

No.	Historias de Usuarios	Puntos de Estimación
1	Conexión a bases de datos y extracción de esquemas relacionales en servidores PostgreSQL.	2
2	Conversión del esquema relacional a la estructura de negocio que soporta esta información en la aplicación.	0.5
3	Generación de los esquemas de mapeo en fichero *.YML para el ORM Propel.	1
4	Mapeo de procedimientos almacenados	6

5	Sincronización de Procedimientos almacenados.	3
6	Sincronización del esquema relacional.	3

Tabla # 2 Estimación de esfuerzo por Historias de Usuario

2.6.3 Fase: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado, cada iteración no debe ser más extensa que tres semanas. Este plan define cuáles HU serán implementadas para cada iteración del sistema y las posibles fechas para estas liberaciones. Al final de la última iteración el sistema estará listo para entrar en producción (26).

2.6.3.1 Plan de iteraciones

El plan de iteraciones especifica las HU que serán implementadas en cada iteración del sistema y las posibles fechas para las liberaciones. Se definieron tres iteraciones para la realización del sistema, las cuales se mencionan a continuación:

- Iteración 1: Esta iteración tiene como objetivo la implementación de la HU número 1,2 y 3 las cuales garantizan la entrada de los parámetros necesarios para comenzar el trabajo de las próximas iteraciones, así como las primeras restricciones fuertes que se deben cumplir.
- Iteración 2: El objetivo de esta iteración es la implementación de las HU número 4, que constituye una de las más complejas a desarrollar.
- Iteración 3: En esta última iteración se implementarán las HU número 5 y 6.

2.6.3.1.1 Plan de duración de las iteraciones

Este plan permite al equipo de desarrollo definir la duración de cada una de las iteraciones, además de mostrar el orden en que serán implementadas las HU.

Iteraciones	Orden de las Historias de Usuario a implementar	Duración total de las Iteraciones
1	Conexión a bases de datos y extracción de esquemas	8 días

	relacionales en servidores PostgreSQL.	
1	Conversión del esquema relacional a la estructura de negocio que soporta esta información en la aplicación.	2 días
1	Generación de los esquemas de mapeo en fichero *.YML para el ORM Propel.	4 días
2	Mapeo de procedimientos almacenados para el Oracle.	24 días
3	Sincronización Procedimientos Almacenados	12 días
3	Sincronización del esquema relacional	12 días

Tabla # 3: Plan de duración de las iteraciones

2.6.3.1.2 Plan de Entregas

A través del plan de entregas clientes y desarrolladores determinan la fecha de entrega de las versiones del producto en las iteraciones estimadas, definiendo el orden en que las HU serán implementadas.

Historia de Usuario	Final 1ra Iteración	Final 2da Iteración	Final 3ra Iteración
Conexión a bases de datos y extracción de esquemas relacionales en servidores PostgreSQL.	x		
Conversión del esquema relacional a la estructura de negocio que soporta esta información en la aplicación.	x		
Generación de los esquemas de mapeo en fichero *.YML para el ORM Propel.	x		
Mapeo de procedimientos almacenados para el Oracle.		x	

Sincronización Procedimientos Almacenados			x
Sincronización del esquema Relacional.			x

Tabla # 4: Plan de Entregas

2.6.3.1.3 Plan de Tareas

El Plan de Tareas permite descomponer cada HU en varias tareas que serán implementadas para dar cumplimiento a la misma, facilitando el trabajo de implementación de los desarrolladores.

Historias de Usuario	Tareas
Conexión a bases de datos y extracción de esquemas relacionales en servidores PostgreSQL.	1. Estudio del catálogo de PostgreSQL. 2. Definición de conexiones, comandos y DAO para PostgreSQL. 3. Operaciones de extracción de información referente a los esquemas relacionales.
Conversión del esquema relacional a la estructura de negocio que soporta esta información en la aplicación.	1. Convertir el esquema extraído a la estructura de entidades del negocio.
Generación de los esquemas de mapeo en fichero *.YML para el ORM Propel.	1. Estudio referente al mapeo para Propel. 2. Definición de una plantilla para estandarizar la forma del mapeo.
Mapeo de procedimientos almacenados para el Oracle.	1. Estudiar las vistas del catálogo de Oracle que devuelvan la estructura de los procedimientos almacenados. 2. Definir funcionalidades en los servicios que brinda el acceso a datos para la extracción de información de los procedimientos almacenados. 3. Definir estructura de entidades del negocio como soporte a la información extraída. 4. Definición y creación de los componentes visuales para el mapeo de procedimientos. 5. Integración de los componentes a la Capa de

	Presentación con un flujo de mapeo de procedimientos. 6. Definir funcionalidades en el gestor de negocio para la configuración del mapeo de procedimientos. 7. Definir la estructura e implementación en el módulo de generación para el mapeo de procedimientos.
Sincronización Procedimientos Almacenados	1. Definir funcionalidades de sincronización para los procedimientos almacenados. 2. Definición y creación de los componentes visuales para la sincronización. 3. Integración de los componentes con un flujo de sincronización del mapeo de procedimientos almacenados en la Capa de Presentación.
Sincronización de esquema relacional	1. Definir funcionalidades de sincronización para los esquemas relacionales. 2. Integración de los componentes creados en la sincronización de procedimientos con un flujo de sincronización para esquema relacional.

Tabla # 5: Plan de Tareas

2.6.4 Fase: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación.

XP propone las tarjetas Clase-Responsabilidad–Colaborador (CRC) como técnica de diseño, la característica más prominente es su simpleza y ductilidad. Una tarjeta CRC establece 3 dimensiones las cuales identifican el rol de un objeto en análisis y/o diseño: nombre de la clase, responsabilidades y colaboraciones (28). Las responsabilidades de una clase son las que

realizan sus métodos. Las colaboraciones de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades.

Clase ProyectoNHiberGen	
Responsabilidades	**Clases relacionadas**
Cargar todos los datos de la Base de Datos. Inicializar las clases. Clase que se encarga de gestionar todo el flujo de trabajo. Se encarga de cargar tablas de una Base de Datos en Oracle, SQL Server o PostgreSQL. Se encarga de probar la conexión configurada. Se utiliza para reconstruir un objeto con nuevas relaciones, es utilizado como una sub función dentro de AgregarRelaciones.	Controladora

Tarjeta CRC: Clase ProyectoNHiberGen

Clase FachadaGeneración	
Responsabilidades	**Tipo de clase**
Clase que se encarga de gestionar todo el flujo de generación de código. Dado un objeto y un lenguaje especificado, genera el código de la clase persistente perteneciente al objeto para el lenguaje especificado. Dado un objeto genera el código del esquema XML o YML perteneciente a la clase persistente generada para el objeto.	Gestora

Tarjeta CRC: FachadaGeneración

Clase GeneradorYML

Responsabilidades	Clases relacionadas
Clase utilizada para generar el esquema YML para la clase generada de un objeto.	FachadaGeneración

Tarjeta CRC: GeneradorYML

Clase Objeto	
Responsabilidades	**Tipo de clase**
Representa las clases mapeadas, contiene la información necesaria para generar una clase persistente. Agrega un nuevo atributo al objeto Recibe como parámetros dos referencias a listas de cadenas en las cuales se almacenan los nombres de las propiedades y los atributos.	Gestora

Tarjeta CRC: Clase Objeto

2.6.4.1 Pruebas

La realización de las pruebas es uno de los aspectos esenciales de la metodología XP, permite detectar desde las primeras iteraciones los errores que pueden surgir, contribuyendo a reducirlos y a obtener un programa con mayor calidad que es capaz de aceptar los cambios.

2.6.5 Fase: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura (27)

2.6.6 Fase: Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema, esto requiere que se satisfagan las necesidades del cliente. Se genera la documentación final del sistema y no se realizan más cambios. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo (27).

2.7 Diccionario de Datos o Catálogo del Sistema

El Catálogo de sistema va a ser utilizado principalmente en la aplicación en el Gestor del Negocio para la extracción de la información referente al esquema relacional con que se va a interactuar dependiendo del gestor seleccionado ya sea PostgreSQL, SQL Server u Oracle. Permitiendo este extraer todos los datos relacionados con el esquema relacional, tablas, atributos de los datos, interrelaciones, además de toda la información referente a los procedimientos almacenados.

2.7.1. Diccionario de Datos

El diccionario de datos se utiliza para llevar un recuento detallado de todas las tablas dentro de la Base de Datos que han sido creadas por el usuario, por el diseñador o por ambos. Contiene todos los nombres y características de atributo de cada una de las tablas en el sistema, metadatos, definidos previamente como datos de los datos.

El diccionario de datos se define como la Base de Datos del diseñador de Base de Datos que registra las decisiones de diseño sobre tablas y sus estructuras (28).

Una herramienta que facilita el control y manejo de la información acerca de datos en el diseño, implementación, operación y expansión de fases de una Base de Datos.

2.7.2. Catálogo del sistema

El Catálogo del sistema es una colección de tablas especiales en una Base de Datos que son propiedad, están creadas y son mantenidas por el propio DBMS. Estas tablas del sistema contienen datos que describen la estructura de la Base de Datos. Las tablas del catálogo del sistema son automáticamente creadas al crear la Base de Datos. Consultando los catálogos del sistema, es posible obtener información acerca de la estructura de una Base de Datos, el acceso

de los usuarios al catálogo es de solo lectura. Los nombres y la organización de las tablas del sistema difieren ampliamente del producto DBMS.

Contenidos del catálogo:

- Tablas: El catálogo describe cada tabla de la Base de Datos, identificando su nombre, su propietario, el número de columnas que contiene, su tamaño, etc.
- Columnas: El catálogo describe cada columna de la Base de Datos, proporcionando el nombre de la columna, la tabla a la que pertenece, su tipo de dato, su tamaño y si están permitidos los NULL.
- Vistas: El catálogo describe cada vista definida en la Base de Datos, incluyendo su nombre, el nombre de su propietario, la consulta que define la vista, etc.
- Privilegios: El catálogo describe cada grupo de privilegios concedidos en la Base de Datos, incluyendo los nombres del donante y el donatario, los privilegios concedidos, el objetivo sobre el cual se han concedido los privilegios, etc.

Usuarios: El catálogo describe a cada usuario autorizado de la Base de Datos, incluyendo el nombre, una forma cifrada de la contraseña del usuario y otros datos (29).

En algunas bibliografías estos términos son utilizados indistintamente en algunas bibliografías por lo que a veces pudiese encontrase que el diccionario de datos está contenido dentro del catálogo del sistema.

2.7.1 Catálogo de Oracle

Oracle cuenta con una serie de tablas y vistas de solo lectura, que conforman una estructura denominada catálogo o diccionario de datos.

El diccionario de datos es creado cuando la Base de Datos se crea (el script de creación de estas tablas y vistas, CATALOG.SQL se encuentra por defecto en el directorio RDBMSADMIN.

Algunos de los datos del catálogo son actualizados continuamente, otros no pueden ser en tiempo real pues esto retardaría el rendimiento de la Base de Datos.

La principal función del catálogo de Oracle es almacenar toda la información de la estructura lógica y física:

- Definiciones de todos los esquemas en la BD (tablas, índices, vistas, clusters, sinónimos, secuencias, procedimientos, funciones, paquetes, triggers, etc.).
- Espacio asignado y actualmente ocupado por un esquema.
- Usuarios de la Base de Datos.
- Privilegios y roles que cada usuario tiene.
- Información sobre quienes han accedido y actualizado esquemas.
- Información general de la Base de Datos (30).

2.7.2 Catálogo de SQL Server

El catálogo del sistema proporciona la información siguiente para las Bases de Datos de SQL Server:

- El número y nombre de las tablas y vistas de una Base de Datos.
- El número de columnas de una tabla o vista, junto con el nombre, el tipo de datos, la escala y la precisión de cada columna.
- Las restricciones definidas en una tabla.
- Los índices y las claves definidas para una tabla.

El núcleo de los catálogos del sistema de SQL Server es un conjunto de vistas que muestran metadatos que describen los objetos de una instancia de SQL Server. Los metadatos son datos que describen los atributos de los objetos de un sistema. Las aplicaciones basadas en SQL Server pueden tener acceso a la información de los catálogos del sistema utilizando lo siguiente:

- Vistas de catálogo. Éste es el método de acceso recomendado.
- Vistas del esquema de información.
- Conjuntos de filas del esquema OLE DB.
- Funciones del catálogo de ODBC.
- Procedimientos almacenados y funciones del sistema (31).

2.7.3 Catálogo de PostgreSQL

Los catálogos del sistema son el lugar donde una gestión de Bases de Datos relacionales del sistema almacena esquemas de metadatos, como la información acerca de las tablas y columnas, etc. Los catálogos del sistema PostgreSQL son las tablas periódicas. Se pueden eliminar y volver a crear tablas, agregar columnas, insertar y actualizar los valores. No es

recomendable cambiar los catálogos del sistema manualmente, existen comandos SQL que encierran estas funcionalidades (32).

Prácticamente todo catálogo contiene alguna referencia a instancias en una o ambas clases. Por ejemplo, PostgreSQL frecuentemente usa firmas de tipo (por ejemplo, de funciones y operadores) para identificar instancias únicas en otros catálogos.

2.8 Interfaz de Usuario

El Paradigma de la Programación Orientada a Componentes es una de las alternativas más utilizadas en la industria de software por las ventajas que este proporciona. La Programación Orientada a Componentes permite una distribución más completa de la funcionalidad en tanto que mantiene una integración estrecha, la actualización o mantenimiento de la aplicación es a nivel de componente por lo que no es necesaria la recompilación de la solución completa sino del componente o los componentes modificados. Posibilita en gran medida la reutilización de componentes, creando componentes con funcionalidades que pueden ser utilizadas en varias partes de la solución. El margen de errores se disminuye en gran medida pues los componentes se someten a rigurosos controles de calidad en los que se validan la posibilidad de errores en la entrada de datos por los usuarios.

2.8.1. Principales interfaces de usuario que incorpora la herramienta.

Una interfaz es un conjunto de comandos o menús a través de los cuales el usuario se comunica con el programa. Esta es una de las partes más importantes de cualquier programa ya que determina que tan fácilmente es posible que el programa haga lo que el usuario quiere hacer. Un programa muy poderoso con una interfaz pobremente elaborada tiene poco valor para un usuario no experto (33).

A continuación se realiza una breve descripción de las principales interfaces de usuario que se incorporaron en la aplicación para facilitar el trabajo con la herramienta:

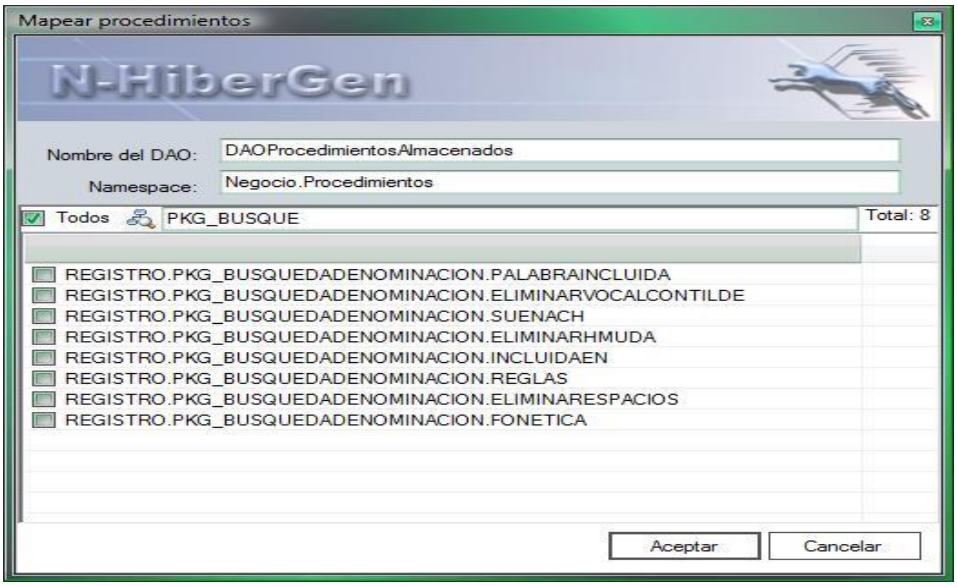


Fig. 8: Extracción de Procedimientos Almacenados

La interfaz brinda la posibilidad de entrar los principales datos de configuración del DAO(El Data Access Object agrupa todas las funcionalidades de los procedimientos almacenados, realiza un encapsulamiento de las funcionalidades del framework NHibernate versión 1.1 la cual contendrá las modificaciones realizadas por los desarrolladores para que permita el mapeo de los procedimientos), dentro de los datos se encontrará el nombre y el namespaces donde se ubicará dentro de este, la lista de procedimientos junto con las funciones de NHibernate que se encapsularán dentro del DAO. Permitirá seleccionar de todos los procedimientos extraídos del Gestor de Base de Datos cuáles se mapearán en el DAO. Además cuenta con un botón Aceptar que procede a extraer el resto de los datos de los procedimientos almacenados.

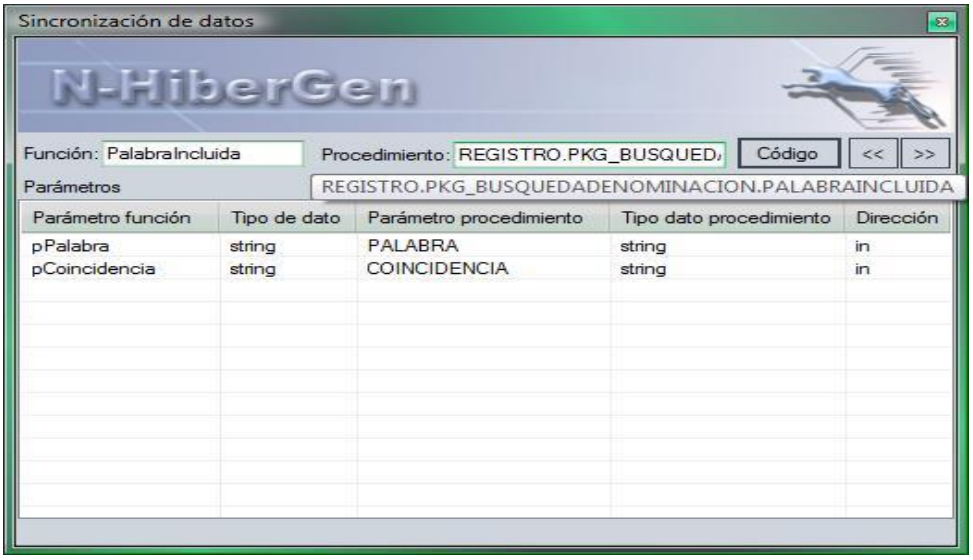


Fig.9: Edición de Procedimiento

Muestra como inicio una vista detallada de los datos de los procedimientos almacenados, permitiendo al usuario editar cambios en la información perteneciente al DAO, tales como nombre de la función y de los parámetros que aparecerán en la función del DAO en caso de que reciba parámetros el procedimiento, cuenta además con un botón código que ofrece una vista previa del mapeo de los procedimientos almacenados y la confección del DAO, utilizando para esto el componente visor de código.

Sincronización del esquema relacional o procedimientos almacenados

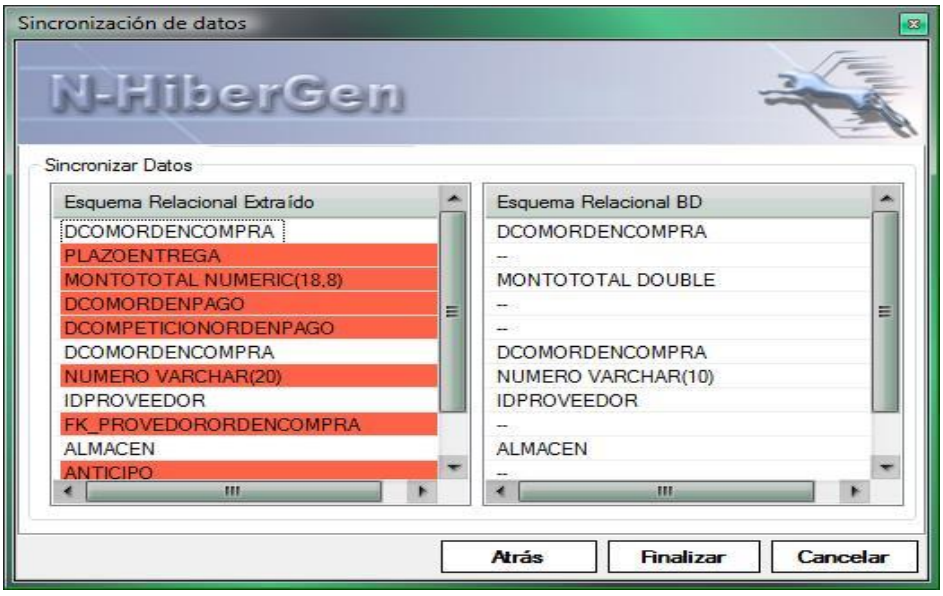


Fig.10: Edición de Procedimiento

La interfaz Sincronización del esquema relacional o procedimientos almacenados, permite sincronizar el esquema relacional o los procedimientos almacenados ya extraídos con la información actual encontrada en la Base de Datos. Muestra dos columnas, la izquierda con la información actual y la columna derecha con la información que difiere de los procedimientos almacenados o el esquema relacional ya extraídos y ofrece como resultado una vista previa de los cambios realizados.

2.9 Conclusiones

En este capítulo se realizó la descripción de la solución propuesta, buscando una mayor comprensión por parte de los desarrolladores que serán los principales usuarios de ésta, se describe la arquitectura empleada y los patrones utilizadas en esta para dar respuesta al problema planteado, se describe el desarrollo de la metodología seleccionada y las distintas técnicas y artefactos que dan soporte al desarrollo del software, se realiza una breve reseña de los catálogos del sistema por el significativo papel que desempeñan éstos y para una vista inicial de la aplicación se describen algunas de las más importantes interfaces de usuario incorporadas a la herramienta.

3 Capitulo 3: Validación de la solución Propuesta

3.1 Pruebas de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software.

Una de las características típicas del desarrollo de software basado en el ciclo de vida es la realización de controles periódicos, normalmente, coincidiendo con los hitos de los proyectos o la terminación de documentos. Estos controles pretenden una evaluación de la calidad de los productos generados (especificación de requisitos, documentos de diseño, etc.) para poder detectar posibles defectos cuanto antes. Sin embargo, todo sistema o aplicación, independientemente de estas revisiones, debe ser probado mediante su ejecución controlada antes de ser entregado al cliente. Estas ejecuciones o ensayos de funcionamiento, posteriores a la terminación del código del software, se denominan habitualmente pruebas (33).

Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. Además, establecen los resultados a alcanzar en correspondencia con la lógica del programa y los datos ingresados. Describen las condiciones generales en las que se deben aplicar las pruebas para obtener los objetivos propuestos. El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones. Es beneficioso que los desarrolladores prueben su producto, pero que no falte la mano de terceras personas que no intervinieron en el proyecto directamente ya que así se detecta una mayor cantidad de fallas.

Aspectos importantes a tener en cuenta al seleccionar las pruebas que se adapten mejor al sistema a probar son: lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan

los procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a Bases de Datos, entre otras observaciones (34).

3.1.1 Pruebas unitarias

Las pruebas unitarias son una de las formas que tenemos de probar pequeñas e individuales porciones de código. A través de ellas se verifica qué módulo o funcionalidad se ejecuta dentro de los parámetros y especificaciones concretadas en documentos tales como los: casos de uso y el diseño detallado, proporcionan un contrato escrito que la porción de código debe cumplir. Permiten detectar efectivamente la inyección de defectos durante fases sucesivas de desarrollo o mantenimiento.

Las pruebas unitarias son una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

Fomentan el cambio: facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

Simplifica la integración: permite llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.

Documenta el código: Las propias pruebas son documentación del código puesto que en estas se permite ver cómo utilizarlo.

Separación de la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.

Los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos (35).

Por qué realizar pruebas unitarias:

• Aseguran calidad del código entregado. Es la mejor forma de detectar errores tempranamente en el desarrollo.

• Ayudan a definir los requerimientos y responsabilidades de cada método en cada clase probada.

• Permiten hacer refactoring tempranamente en el código. No es necesario todo un ciclo de integración para hacer refactoring en la aplicación, basta con ver cómo se comporta un caso de prueba para hacer refactoring unitario sobre la clase que se está probando en cuestión.

• Permiten incluso hacer pruebas de estrés tempranamente en el código. Por ejemplo un método que realice una consulta SQL que exceda los tiempos de aceptación es posible optimizarla antes de integrar con la aplicación.

• Permiten encontrar errores o bugs tempranamente en el desarrollo. Y está demostrado que cuanto más temprano se corrijan los errores, menos costará corregirlos (36).

Los dos grandes grupos de pruebas unitarias que existentes son: las pruebas de Caja Negra y las pruebas de Caja Blanca.

3.1.2 Pruebas de Caja Blanca

Las pruebas de caja blanca realizan un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones, bloques, etc., en los que existen errores.

Cuando se pasan casos de prueba al programa que se está probando, es conveniente conocer qué porcentaje del programa se ha ejecutado, de manera que estemos próximos a asegurar que todo en él es correcto (evidentemente, es imposible alcanzar una certeza del 100%) (37).

Para esta prueba se consideran tres importantes puntos:

- Conocer el desarrollo interno del programa, determinante en el análisis de coherencia y consistencia del código.
- Considerar las reglas predefinidas por cada algoritmo.
- Comparar el desarrollo del programa en su código con la documentación pertinente.

Las pruebas de caja blanca se realizan sobre las funciones internas de un módulo en concreto. Entre las técnicas usadas se encuentran: la cobertura de caminos, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles

En la siguiente tabla se representan las pruebas de Caja Blanca.

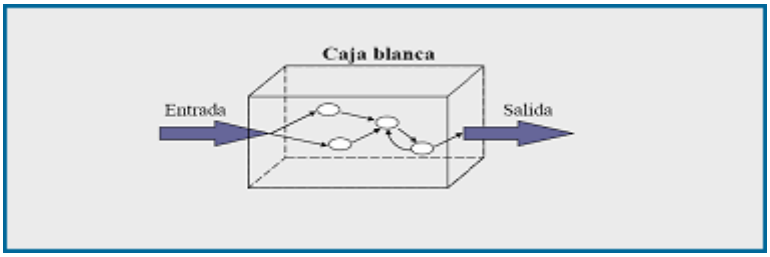


Fig. 10: Representación de pruebas de Caja Blanca

Tipos de prueba de caja blanca:

- Prueba de Condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- Prueba de Flujo de Datos: Se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- Prueba de Bucles: Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.
- Prueba del Camino Básico: Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de pruebas a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática.

La técnica del camino básico permite obtener una medida de la complejidad lógica del código de cada método, programa o módulo dado. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes que existen en la codificación por los cuales puede circular el flujo de control. Es además una de las más eficientes en cuanto a cobertura de código, pues logra que se ejecuten todos los bucles en sus límites operacionales (38).

Los pasos que se siguen para aplicar esta técnica son:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.

- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba obtenidos del conjunto básico, garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, este puede representarse por un grafo de flujo en el cual:

- Cada nodo del grafo corresponde a una o más sentencias de código fuente.
- Todo segmento de código de cualquier programa se puede traducir a un grafo de flujo.
- Se calcula la complejidad ciclomática del grafo.

Para construir el grafo se debe tener en cuenta la notación para las instrucciones.

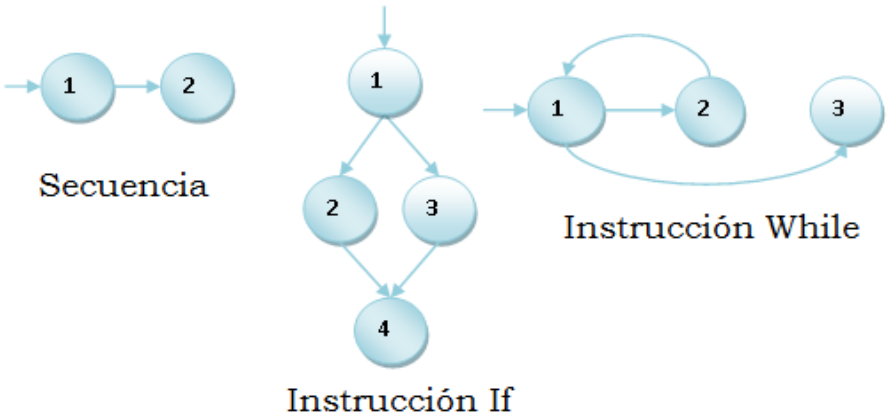


Fig. 11: Notación de grafos de flujo para las instrucciones: Secuenciales, If y While.

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración, su comprensión y brindan información para confirmar que el trabajo se está haciendo adecuadamente.

Componentes del grafo de flujo:

Nodo: son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento.

Regiones: son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

Para realizar la prueba de caja blanca específicamente la prueba del camino básico, es necesario, calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar.

Cálculo de la complejidad ciclomática.

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 da un límite superior para el número de pruebas que se deben realizar, para asegurar que se ejecute cada sentencia al menos una vez.

Para efectuar el cálculo de la complejidad ciclomática del código es necesario tener varios parámetros como son la cantidad total de aristas del grafo, cantidad total de nodos para la siguiente fórmula:

V (G) = (A– N) + 2

Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

Se puede usar también:

V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = R

Siendo “R” la cantidad total de regiones, para cada formula “V (G)” representa el valor del cálculo.

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico.

Para realizarlos es necesario cumplir con las siguientes exigencias:

- Descripción**:** Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.

- Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.
- Entrada**:** Se muestran los parámetros que entran al procedimiento.
- Resultados Esperados: Se expone resultado que se espera que devuelva el procedimiento.

Para realizar la prueba de caja blanca, específicamente la prueba del camino básico es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar de la clase controladora ProyectoNHiberGen:

```
private void EliminarObjeto (string pNombreObjeto, List<Objeto> pColObjetos)
    {
        for (int i = 0; i < pColObjetos. Count; i++)//1
        {
            if (pColObjetos[i].NombreObjeto == pNombreObjeto)//2
            {
                if (pColObjetos[i] is ObjetoHerencia)//3
                {
                    foreach (Objeto hijo in ((ObjetoHerencia) pColObjetos[i]).ObjetosHijos)//4
                    {
                        colObjetos.Add(hijo);//5
                    }
                }
                pColObjetos.RemoveAt(i);//6
                i--; //6
                estaSalvado = false;//6
            }
            else
            {
```

```
            for (int j = 0; j < pColObjetos[i].Atributos.Count; j++)//7
            {
                Atributo a = pColObjetos[i].Atributos[j];//8
                if (a is AtributoRelacionUnoMucho
                    && ((AtributoRelacionUnoMucho)a).NombreObjRelacion == pNombreObjeto)//9
                {
                    pColObjetos[i].Atributos.RemoveAt(j);//10
                    j--;//10
                    estaSalvado = false;//10
                }
                if (a is AtributoRelUnoUnoInd
                    && ((AtributoRelUnoUnoInd)a).NombreObjRelacion == pNombreObjeto)//11
                {
                    pColObjetos[i].Atributos.RemoveAt(j);//12
                    j--;//12
                    estaSalvado = false;//12
                }
                if (a is AtributoRelUnoUnoDep
                    && ((AtributoRelUnoUnoDep)a).NombreObjRelacion == pNombreObjeto)//13
                {
                    pColObjetos[i].Atributos[j] = new AtributoCampo(((AtributoRelUnoUnoDep)
.ColumnaBD, true);//14
                    pColObjetos[i].Atributos[j].Propiedad = a.Propiedad;//14
                    estaSalvado = false;//14
                }//15
            }
```

```
            if (pColObjetos[i] is ObjetoHerencia)//16
                EliminarObjeto(pNombreObjeto,
((ObjetoHerencia)pColObjetos[i]).ObjetosHijos);//17
        }//18
    }//18
}//19
```

En la siguiente figura se muestra el grafo del flujo asociado al código anterior.

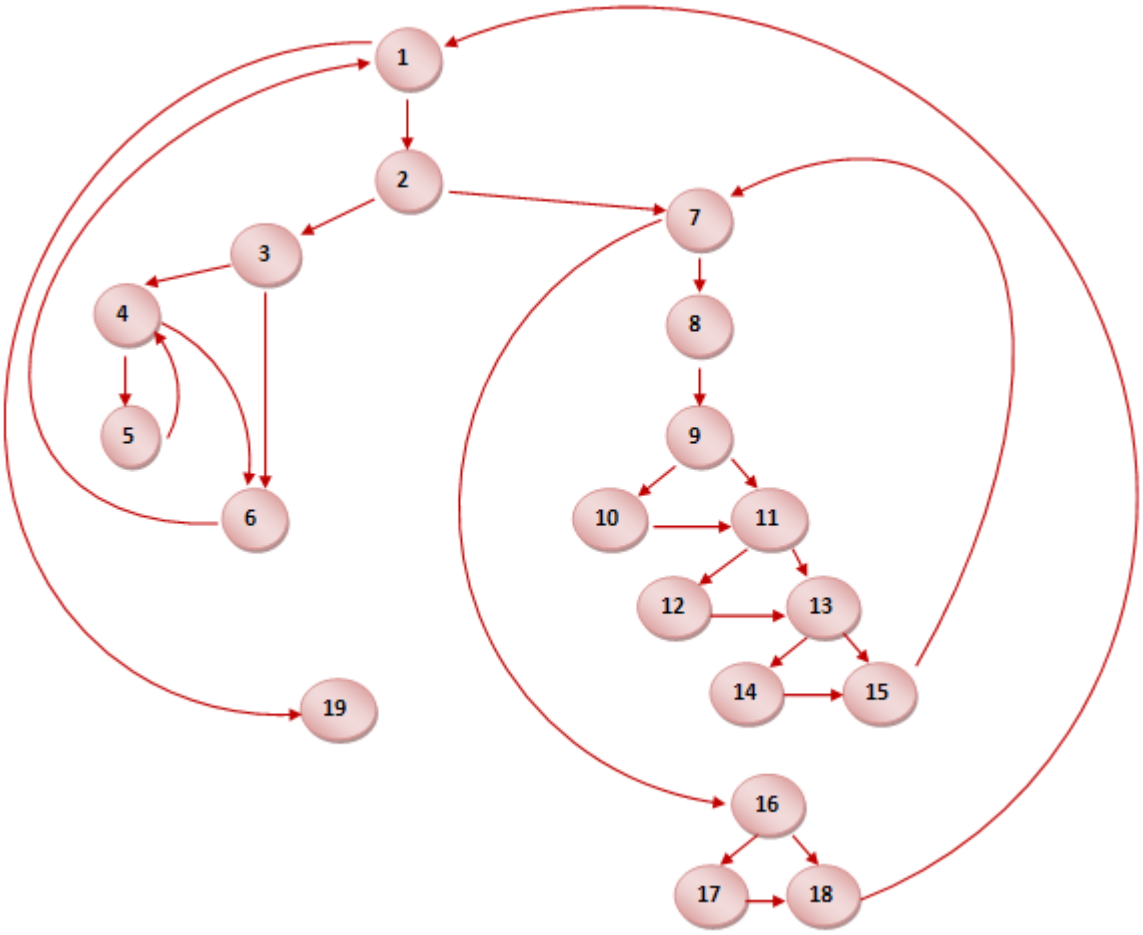


Fig.12: Grafo del código de Caso de Prueba.

Cálculo de la complejidad ciclomática variante 1:

V (G) = (A– N) + 2

V (G) = (27 – 19) + 2

V (G) = 10

Cálculo de la complejidad ciclomática variante 2:

V (G) = P + 1

V (G) = 9 + 1

V (G) = 10

Cálculo de la complejidad ciclomática variante 3:

V (G) = R

V (G) = 10

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede plantear que la complejidad ciclomática del código es 10, lo que significa que existen diez posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo.

En la siguiente tabla se muestran los caminos básicos.

Número	Caminos básicos
1	1-19
2	1-2-3-4-6-1-19
3	1-2-7-16-18-1-19
4	1-2-3-4-5-4-6-1-19
5	1-2-7-16-17-18-1-19
6	1-2-7-8-9-11-13-15-7-16-18-1-19
7	1-2-7-8-9-11-13-15-16-17-18-1-19
8	1-2-7-8-9-11-12-13-15-16-18-1-19
9	1-2-7-8-9-11-13-15-7-16-17-18-1-19
10	1-2-7-8-9-10-11-12-13-14-15-7-16-17-18-1-19

A cada camino básico se le realiza un caso de prueba el cual va a consistir en hacer una descripción del método y dar la condición de ejecución, luego se le pasan variables y se esperan los resultados.

3.1.3 Pruebas utilizando NUnit

Para la validación de la aplicación se utilizó el framework de software libre NUnit, que se encarga de realizar pruebas a módulos o segmentos de código para verificar que funcionen apropiadamente.

NUnit es una herramienta que se encarga de analizar ensamblados generados por .NET, interpretar las pruebas inmersas en ellos y ejecutarlas. Utiliza atributos personalizados para interpretar las pruebas y provee además métodos para implementarlas. En general, NUnit compara valores esperados y valores generados, si estos son diferentes la prueba no pasa, caso contrario la prueba es exitosa. NUnit carga en su entorno un ensamblado y cada vez que lo ejecuta, o mejor, ejecuta las pruebas que contiene, lo recarga. Esto es útil porque se pueden tener ciclos de codificación y ejecución de pruebas simultáneamente, así cada vez que se compile no tiene que volver a cargar el ensamblado al entorno de NUnit si no que este siempre obtiene la última versión del mismo. NUnit ofrece una interface simple que informa si una prueba o un conjunto de pruebas fallaron, pasaron o fueron ignorados (41).

En la implementación de estas pruebas se definió un proyecto en el cual se incluyen los ensamblados de la solución y el framework NUnit, creando una clase para las pruebas llamada CasoPrueba donde se implementaron las pruebas realizadas a la aplicación.

Como punto de inicio a las pruebas realizadas se definió el siguiente, cargando un caso de prueba previamente creado para las pruebas que se realizaron.

```
[SetUp]
public void InicioPrueba()
{
    Singleton.AbrirProyecto("CasoPrueba.nhgen");
    proyPrueba = Singleton.ProyectoCAD;
}
```

Partiendo de los datos cargados inicialmente se le realizaron pruebas al gestor principal de la aplicación.

```
[Test]
public void ObjetoPorNombre()
{
    Assert.IsNotNull(proyPrueba.ObjetoPorNombre("Departamento"), "Valor no esperado");
}
```

Este método verifica la existencia de un objeto dentro de un proyecto, se espera la devolución del mismo, según el caso de prueba implementado, y por el valor generado la prueba fue exitosa.

```
[Test]
public void TieneRelacionUnoUno()
{
    Objeto asignatura = proyPrueba.ObjetoPorNombre("Asignatura");
    Objeto departamento = proyPrueba.ObjetoPorNombre("Departamento");
    List<Objeto> relacionesUnoUno = new List<Objeto>();
    proyPrueba.TieneRelacionUnoUno(asignatura, departamento, ref relacionesUnoUno);
    Assert.AreEqual(0, relacionesUnoUno.Count, "Valor no esperado");
}
```

Este método verifica si existe relación de uno a uno entre dos objetos, devolviendo una referencia con una lista de las posibles relaciones que pueden existir de uno a uno incluso con los hijos del segundo objeto en caso de que existan objetos que hereden de él, según el caso de prueba implementado, se espera como respuesta que no exista relación entre los objetos, por tanto, la lista debe tener 0 como cantidad de elementos, y por los resultados generados la prueba resultó satisfactoria.

```
[Test]
public void CambiarNombreObjeto()
{
    proyPrueba.CambiarNombreObjeto("EDepartamento", "Departamento");
    proyPrueba.CambiarNombreObjeto("EAsignatura", "Asignatura");
    proyPrueba.CambiarNombreObjeto("EPersona", "Persona");
}
```

Este método cambia el nombre de un objeto, recibiendo como parámetro primero el nombre nuevo que se le asignará al objeto y el nombre que tiene en ese momento, según el caso de prueba implementado, se espera como respuesta el cambio de todos los nombres de objeto, y para la respuesta generada la prueba resultó satisfactoria.

```
[Test, ExpectedException(typeof(Exception))]
public void CambioNombrePropiedad()
{
    proyPrueba.CambioNombrePropiedad("Departamento", "NombreDepartamento", "P_Nombredepartamento");
    proyPrueba.CambioNombrePropiedad("Departamento", "Asignaturas", "List_Asignatura");
    proyPrueba.CambioNombrePropiedad("Departamento", "Profesores", "List_Profesor");
    proyPrueba.CambioNombrePropiedad("Departamento", "decimal", "P_Iddepartamento");
}
```

Este método cambia el nombre de una propiedad de un objeto, recibiendo por parámetro primero el nombre del objeto al que se le quiere cambiar el nombre de la propiedad, luego el nombre que se le intercambiará a la propiedad y por último el nombre que tiene la propiedad en ese momento, antes de cambiar se verifica que el nuevo nombre de propiedad sea válido, teniendo en cuenta una serie de parámetros o requisitos; como respuesta de esta prueba se espera una excepción (Exception) porque el nuevo nombre de la propiedad “P_Iddepartamento” no es válido, y como la respuesta generada fue la esperada la prueba resultó satisfactoria.

```
[Test]
public void MostrarCodigoClaseCSharp()
{
    string cadenaClase2003 = "";
    string cadenaClase2005 = "";
    string cadenaInterface2003 = "";
    string cadenaInterface2005 = "";
    string[] reservadas = new string[0];
    proyPrueba.GenerarCodigoCSharp("Asignatura",
                                   ref cadenaClase2003,
                                   ref cadenaClase2005,
                                   ref cadenaInterface2003,
                                   ref cadenaInterface2005,
                                   ref reservadas);
    Assert.IsTrue(visor.MostrarCodigoClase(cadenaClase2005,
                                           reservadas),
                  "Valor no esperado");
}
```

Este método verifica que el componente UCVisorCodigo muestre el código de la clase, por el método MostrarCodigoClase que recibe una cadena con el código de la clase a mostrar y las palabras reservadas adicionales que se puedan mostrar, según el caso de prueba implementado se espera respuesta positiva, y para el valor generado la prueba resultó satisfactoria.

```
[Test]
public void MostrarEsquemaXML()
{
    string cadenaXML = "";
    string[] reservadasXML = new string[0];
    proyPrueba.GenerarCodigoXML("Asignatura",
                                ref cadenaXML,
                                ref reservadasXML);
    Assert.IsTrue(visor.MostrarCodigoXML(cadenaXML,reservadasXML),
                  "Valor no esperado");
}
```

Este método verifica que el componente UCVisorCodigo muestre el código del esquema XML, por el método MostrarCodigoXML que recibe una cadena con el código del esquema XML a mostrar y las palabras reservadas adicionales que se puedan mostrar, según el caso de prueba implementado se espera respuesta positiva, y para el valor generado la prueba resultó satisfactoria.

```
[Test]
public void MostrarDiagramaClases()
{
    JerarquiaClase jerarquiaClase
                   = CrearJerarquiaClases("Usuario");
    Assert.IsTrue(diagrama.MostrarJerarquiaClase(jerarquiaClase));
}
```

Este método verifica que el componente UCDiagramaClases muestre el diagrama de clase de un objeto, por el método MostrarJerarquiaClase que recibe como parámetro una jerarquía de la clase que se quiere mostrar, según el caso de prueba implementado se espera respuesta positiva.

Después de aplicadas las pruebas se puede observar en la figura que se muestra a continuación, donde aparece la interfaz de NUnit, la correcta ejecución de las pruebas aplicadas a la aplicación y los resultados satisfactorios de las mismas.

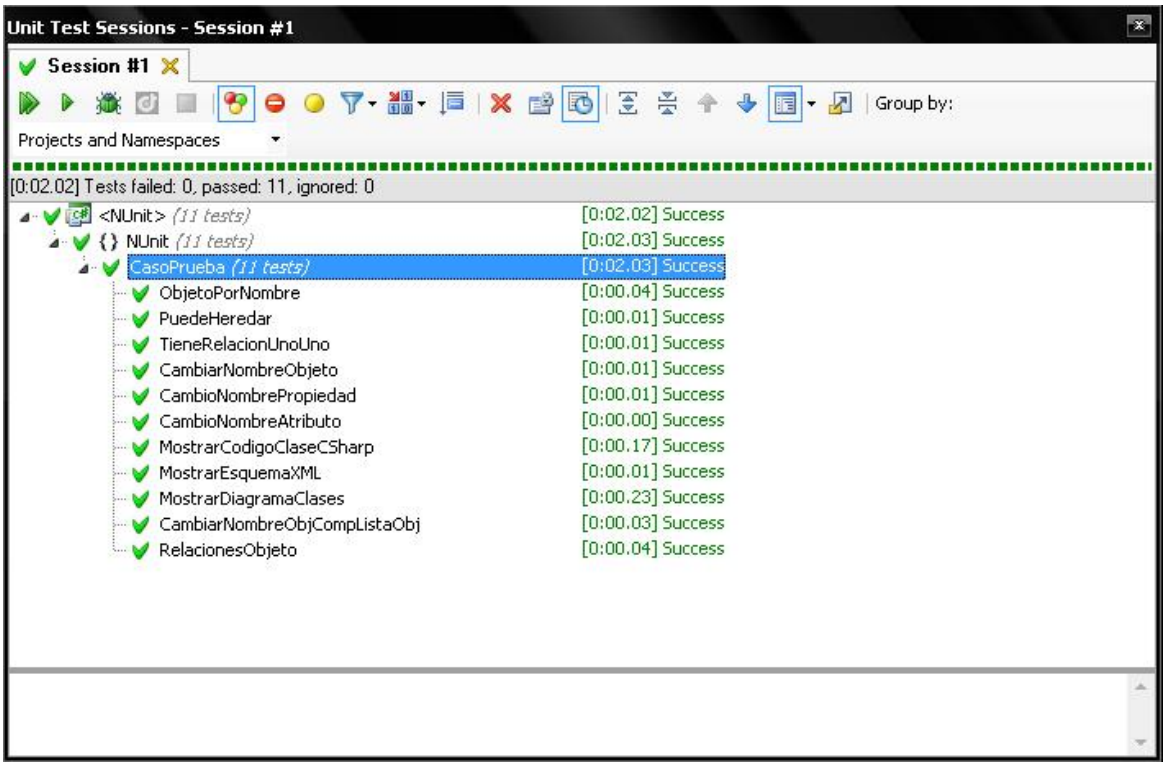


Figura 21. Resultados pruebas con NUnit.

3.2 Conclusiones

En este capítulo se realizaron pruebas de software mediante el método de caja blanca, utilizando para esto la técnica del camino básico, la cual permitió obtener una medida de la complejidad lógica del diseño, se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática para obtener un conjunto de caminos independientes. En la aplicación se utilizó el framework de software libre NUnit, para hacer pruebas a módulos o segmentos de código y garantizar que funcionen correctamente, todas estas pruebas se llevaron a cabo para garantizar a calidad de la herramienta desarrollada y con esto mitigar la cantidad máxima de errores que puedan existir.

Conclusiones Generales

Con la realización y terminación de este trabajo se llega a las siguientes conclusiones:

- Con el estudio de los sistemas generadores de ficheros de mapeo para la persistencia de esquemas mediante el frameworks NHibernate, Hibernate y Propel se logró un mayor conocimiento de las funcionalidades que ofrecen estas herramientas y sirvió de ejemplo para el desarrollo de la aplicación.
- Se definió basado en los principios y cualidades del tipo de software, una arquitectura que guió todo el proceso de desarrollo del software.
- Fueron estudiados los catálogos de sistemas para los Gestores de Bases de Datos PostgreSQL, SQLServer y Oracle, debido a la importancia de estos en la aplicación.
- Se definió la metodología de desarrollo de software a utilizar basado en las características del proyecto, así como las herramientas y tecnologías para el desarrollo del software.
- Se logró la implementación de las nuevas funcionalidades para dar respuesta al problema planteado, mejorando la herramienta existente.
- Para garantizar la calidad del sistema se utilizaron las pruebas de caja blanca, apoyándose principalmente en la herramienta NUnit y usando el método del camino básico.

Recomendaciones

Se recomienda para futuras iteraciones y versiones del sistema propuesto:

- Migrar la solución propuesta a una plataforma libre, preferiblemente Mono, para lograr una mayor reutilización de los componentes generados.
- Se propone que se realice el mapeo de procedimientos almacenados para los frameworks Hibernate y Propel con los gestores PostgreSQL y SQLServer.
- Incluir nuevos framework de persistencia como doctrine.
- Realizar las plantillas necesarias para el uso de XSLT, para la generación de clases e interfaces en los distintos lenguajes de programación.

Bibliografía

1. **Eguíluz, Javier.** librosweb.es. *CSS Avanzado.* [En línea] http://librosweb.es/css_avanzado/capitulo5.html.

2. **Lazarte, Daniel.** Perú Architect's and Develoepers Microsoft Technology. *.Net Framework.* [En línea] 18 de octubre de 2007. http://peruti.wordpress.com/category/net-framework/.

3. **Grupo de Investigaciones en Ingenieria de Software.** *Tertulia de Ingenieria de Software.* 15 de septiembre del 2005.

4. **Markiewicz, Marcus E. y de Lucena, Carlos J.P.** [En línea] http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html.

5. **Gomez, Jesus.** Desarrollo y demas cosas. *Hibernate y JDeveloper.* [En línea] 5 de mayo de 2009. http://jesusgomez.blogspot.com/2009/08/hibernate-y-jdeveloper.html.

6. **Castellano, Raúl.** *Motores de Persistencia.* 2008.

7. **Larman, C.** *Persistencia.* s.l. : Addison Wesley, 2002.

8. Motores de persistencia. [En línea] http://petra.euitio.uniovi.es/~i1643233/MotoresDePersistencia-v02.pdf.

9. Tutorial IT blog. *NHibernate para .NET.* [En línea] 2 de Noviembre de 2008. http://www.conexionit.com/blog/herramientas/nhibernate-para-net.html.

10. msdn. *Información general de Entity Framework.* [En línea] http://msdn.microsoft.com/es-es/library/bb399567%28v=VS.100%29.aspx.

11. **Mendoza, María A.** Informtizate. *Metodología de Desarrollo de Software.* [En línea] 7 de Junio de 2004. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

12. wigahluk.wordpress.com. [En línea] 26 de Junio de 2007. http://wigahluk.wordpress.com/2007/06/26/entre-la-xp-y-el-rup/.

13. proyectosagiles.org. *SCRUM.* [En línea] http://www.proyectosagiles.org/que-es-scrum.

14. pymecrunch.com. *Scrum:metodologia agil para tus proyectos.* [En línea] 2 de abril de 2008. http://pymecrunch.com/scrum-metodologia-agil-para-tus-proyectos.

15. **Villahermosa, Ramon.** rvillahermosa.blogspot.com. *Herramientas ORM.* [En línea] 1 de octubre de 2007. http://rvillahermosa.blogspot.com/2007/10/herramientas-orm.html.

16. MyGenerator. [En línea] http://www.mygenerationsoftware.com/portal/default.aspx.

17. GenWise. *GeneratorProject.* [En línea] http://www.genwise.com/Products/WhatIsGenWise/GenWiseProductDetails/tabid/145/Default.aspx.

18. **Leyet Fernández, Osmar y Rodriguez Lorenzo, Iosmel.** *Desarrollo de una herramienta generadora de ficheros de mapeo para la persistencia de objetos relacionales basada en NHibernate.* Ciudad Habana : s.n., junio del 2008.

19. The ObjectMapper .NET Project. *The ObjectMapper .NET .* [En línea] http://blog.objectmapper.net/.

20. [aut. libro] Erick Gamma, y otros. *Design Patterns.* s.l. : Grady Booch.

21. PmWiki. *Patron Facade.* [En línea] http://petra.euitio.uniovi.es/~i6950404/wiki/pmwiki.php?n=Tema5.PatronFacade.

22. **Welicki, León.** msdn. *El Patrón Singleton .* [En línea] http://msdn.microsoft.com/es-es/library/bb972272.aspx.

23. msdn. *Documentación de Visual Studio .* [En línea] http://msdn.microsoft.com/es-es/library/ms269115(v=VS.80).aspx.

24. **Recio, Francisco y Provencio, David.** *.Net Framework.* [En línea] 10 de Diciembre de 2003. http://www.desarrolloweb.com/articulos/1328.php.

25. WebEstilo. *Manual Asp.Net.* [En línea] http://www.webestilo.com/aspnet/aspnet00.phtml.

26. **Campos, Mauricio.** *Extreme Programming.* [En línea] Julio de 2004. http://apit.wdfiles.com/local--files/start/02_apit_xp_conceptos.pdf..

27. **Letelier, Patricio y Penadés, Mª Carmen.** Laboratorio de Sistemas de Información. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* [En línea] 2004. http://www.dsic.upv.es/asignaturas/facultad/lsi/doc/MetodologiasAgilesyExtremeProgramming.doc..

28. **Casas, Sandra y Reinaga, Héctor.** *Identificación y Modelado de Aspectos Tempranos dirigido por Tarjetas de Responsabilidades y Colaboraciones.* [En línea] 5 de mayo de 2009. http://espanol.oocities.com/profeprog2/INVPAPER25.pdf..

29. **Rob, Peter y Coronel, Carlos.** *Sistemas de Base de Datos.* s.l. : Nieto Impresores S.A de C.V, 2006.

30. El Catálogo del Sistema. [En línea] http://cursos.itam.mx/akuri/2002/S12002/BasesDeDatos/CHAPTER6.PDF.

31. **Pousa, Javier.** MYGNET. *Introducción al Catálogo de Oracle.* [En línea] 13 de Diciembre de 2005. http://www.mygnet.net/articulos/oracle/introduccion_al_catalogo_de_oracle.246.

32. msdn. *Consulta y modificación de datos (motor de base de datos).* [En línea] Noviembre de 2008. http://msdn.microsoft.com/es-es/library/bb510440(v=SQL.105).aspx.

33. PostgreSQL. *Chapter 44. System Catalogs.* [En línea] http://www.postgresql.org/docs/8.3/static/catalogs.html.

34. **Myers, B. A.** Interface Builders. [En línea] 1996.

35. El Rincón del Vago. *Diseño de la interfaz de usuario: Pruebas del software.* [En línea] http://www.google.com.cu/imgres?imgurl=http://html.rincondelvago.com/000225351.jpg&imgrefurl=http://html.rincondelvago.com/diseno-de-la-interfaz-de-usuario_pruebas-del-software.

36. **Tran, Eushiuan.** *Validacion.* Carnegie Mellon University : s.n., 2004.

37. **Pressman, Roger.** *Ingenieria de Software.* 2002.

38. **Rodriguez, Jorge.** *Pruebas unitarias.* mrzo 2002.

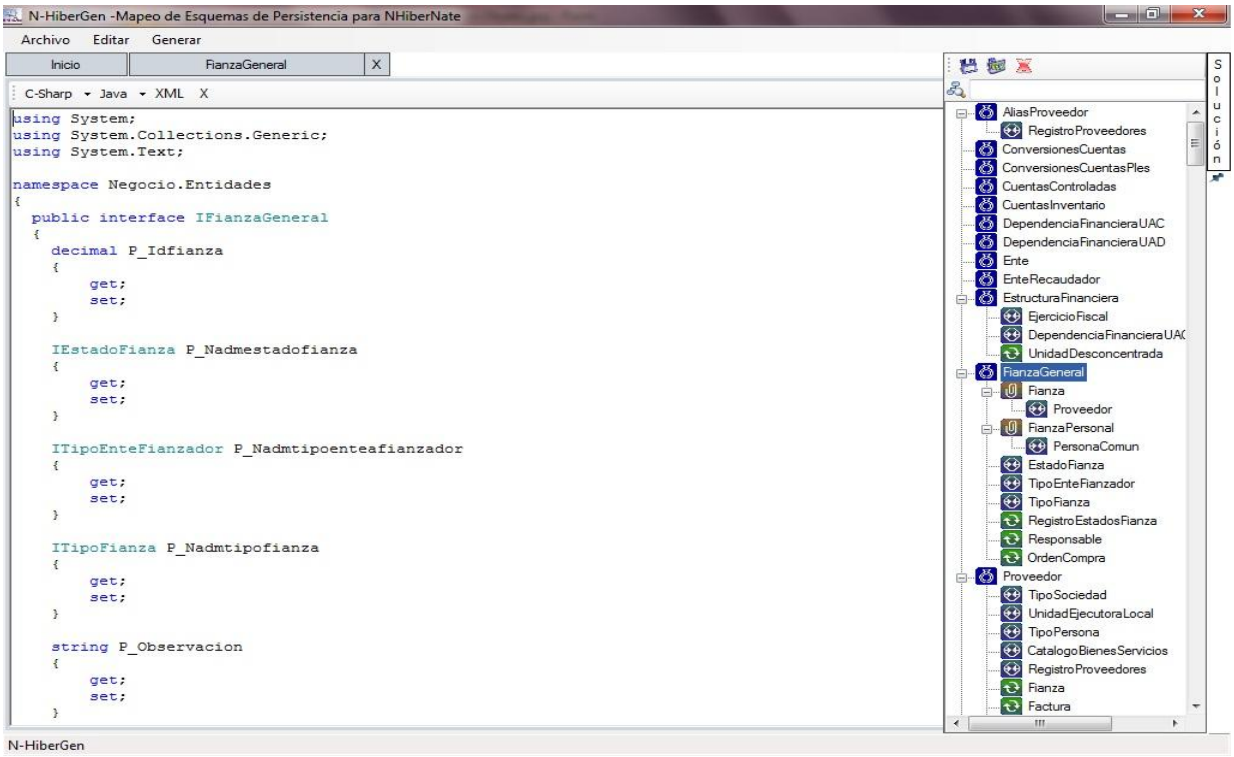
39. **Polo, Macario.** *Mantenimiento Avanzado de Sistemas de Informción.* Ciudad Real : s.n., 2002. 13071.

40. **Lam, Dra. Rosa María.** Instituto de Hepatología e Inmunología. [En línea] http://eva.uci.cu/file.php/63/Bibliografia_del_tema_2/Como_escribir_un_proyecto_de_investigacion.pdf

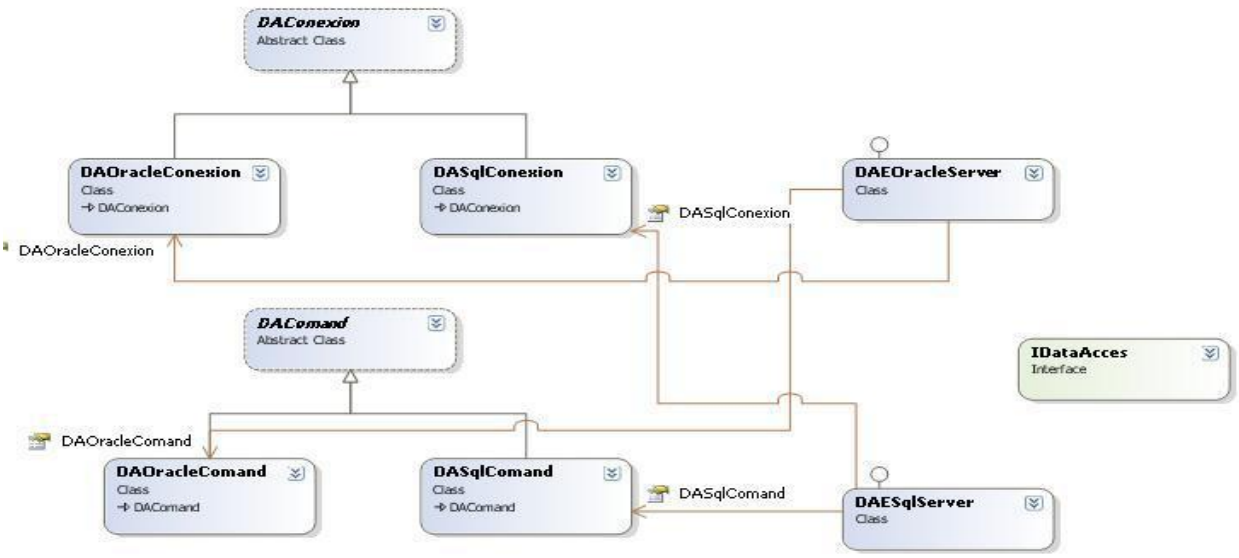
41. EL Rincón del Vago. *Pruebas de Software.* [En línea] http://html.rincondelvago.com/prueba-de-software.html .

42. **Márques Alpízar, Yaimí y Valdez Echavarrí, yenni.** *Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico.* 2008.

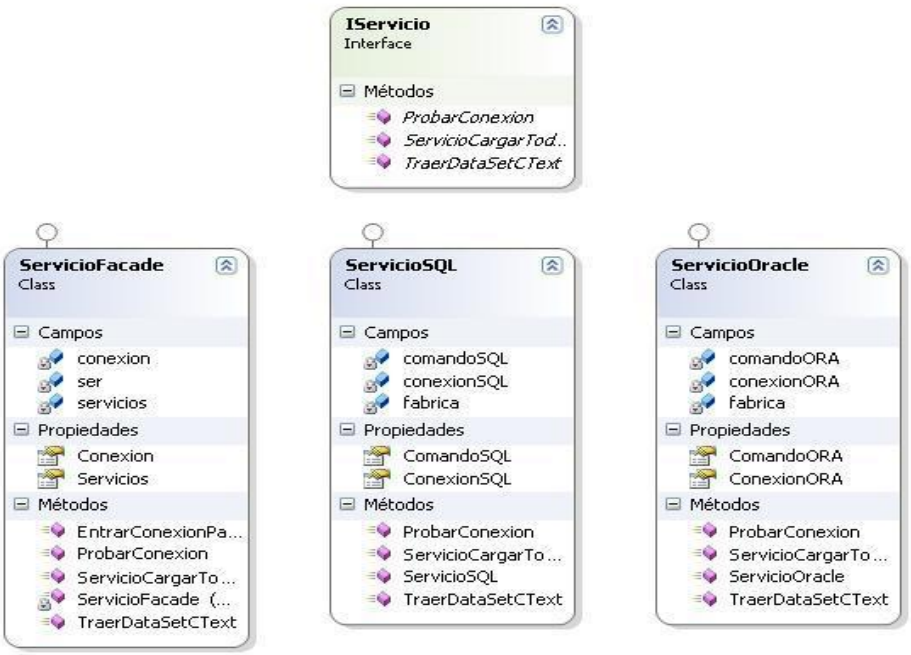
Anexos



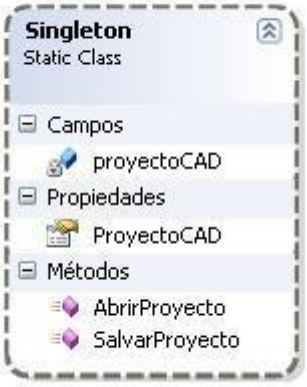
Anexo # 1 Vista previa generada por el sistema.



Anexo # 2: Implementación del patrón Fabrica Abstracta



Anexo # 3: Implementación del patrón Fachada



Anexo 4: Implementación del patrón Solitario.