# UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

## Facultad 3


# *Título: Análisis y Diseño del Subsistema Configuración de Datos de un sistema para la generación de reportes dinámicos.*

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

**Autores**: *Lisandra Proenza Romero*
*Ariana Ramírez Sierra*

**Tutor:** *Ing. Osmany Becerra González*

**Co-Tutor:** *Ing. Carlos Yasmany Hidalgo García*

Ciudad de la Habana
Junio del 2009

*"Se puede albergar un sueño durante años y años, y convertirlo en realidad de repente. Sé paciente. Te pasará, tarde o temprano: ¡la vida te abrirá la puerta, y te permitirá entrar y dar una gran fiesta!"*

*Louis Brown*

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 ____ días del mes de _____ del año _____.

_____ Nombre del Autor

_____ Nombre del Autor

_____ Nombre del Tutor

# *Agradecimientos*

## *De Lisandra*

*A mis padres por su preocupación, su apoyo, por quererme tanto darme lo mejor de ellos, por saber guiarme sin presionarme, gracias por confiar en mí.*

*A mi hermana por quererme tanto y estar siempre conmigo.*

*A mi compañera de tesis, Ariana por soportarnos una a la otra en la confección de esta tesis, por su amistad sin reservas durante estos 5 años de carrera y por la confianza depositada para que saliera este trabajo.*

*A mi tutor Osmany y Co-Tutor Carlos Yasmany por apoyarnos y dedicarnos siempre un tiempito para la elaboración de la tesis.*

*A Enrique y a Yudel por la paciencia, la ayuda, su amistad y la contribución con este trabajo diploma.*

*A todos mis compañeros de aula y estudio en estos cinco años de constante intercambio, Kenia, Yaidel, Any, Adrian, Magelis, Roberto, Osmar, en fin a todos no tengo que ponerlos aquí ellos saben quienes son, nunca los olvidaré.*

*A mis amigos por estar ahí conmigo, por apoyarme siempre, por ser incondicionales y cuidar siempre de mí.*

*A mi amigo Juanki, le doy gracias a la vida por darme la oportunidad de haberte conocido, gracias por tu preocupación, tu atención, tu cariño constante, te quiero mucho.*

*A Jairol, el niño más maravilloso e insoportable que he conocido, gracias por tu amistad y por ser siempre el mismo, sabes que te quiero con la vida.*

*A Gustavo, el más intranquilo de todos, nunca te olvidaré.*

*A las muchachitas del apto por aguantarnos en los momentos de stress, por siempre buenas amigas, por estar siempre con nosotras en todo momento, Ivón, Idaliana, Sahily, Malena.*

*A Yoney y Marcel por su preocupación constante, y el cariño brindado durante todo este tiempo.*

*Aunque en los finales no se encontraban aquí siempre tuve su apoyo desde lejos,*

*a mi gran amiga Yami, Arianna, y al Negro Bladimir.*

*A mis amigas del pre-universitario que desde la distancia nos apoyamos siempre, Lili, Raiza, Dagneris, gracias por estar siempre ahí.*

## *Dedicatoria*

## *De Lisandra*

*A mis padres por ser tan especiales, por confiar en mí y estar conmigo en todo momento, en las buenas y en las malas, apoyándome y dándome la fuerza que he necesitado en todo este tiempo, los quiero mucho.*

*A mi hermana por sus consejos, su optimismo y su ayuda incondicional, Liu siempre te tengo presente.*

*A todos mis familiares que me han apoyado y querido, mis tíos, primos, mi cuñado Jose, gracias por todo y tanto.*

# *Agradecimientos*

## *De Ariana*

*A mi tía por confiar en mí y por estar siempre a mi lado, por guiarme por este camino y por darme su amor y su comprensión, sabes que sin la confianza que depositaste en mí, sin tu apoyo incondicional, sin tu dureza en algunos momentos conmigo nunca hubiera podido llegar a este momento, se que está de más decirlo pero gracias y te quiero mucho.*

*A mi compañera de tesis Lisandra por haber sido mi amiga desde el primer momento, por soportarnos una a la otra en los momentos de stress y por confiar en que juntas podríamos lograrlo, gracias, si tuviera que hacer otra tesis no dudes que te escogería a ti.*

*A mi buena amiga Beatriz por su lealtad, su cariño, su ayuda, su apoyo y su confianza en mí.*

*A Idolidia por quererme tanto y ayudarme, eres especial y me siento honrada de que estés en mi vida y la de mi tía.*

*A Asnel por ser un gran amigo, por darme su confianza y apoyarme tanto aun estando lejos.*

*Al Negro Juan por ser tan especial, por estar siempre a mi lado y quererme tanto.*

*A mi hermano Raine por preocuparse por mí, quererme y apoyarme.*

*A Yudel primero por ser un gran amigo, aunque a veces duro y cruel, eso te hace único y segundo por ayudarme no solo en la tesis sino a lo largo de estos 5 años, gracias por tu tiempo, tu paciencia y tu amistad.*

*A Enrique sin el cual no hubiese sido posible la realización de esta tesis, gracias por dedicarnos tanto tiempo y esfuerzo y también por tu paciencia.*

*A Yoney por ayudarme, apoyarme y soportarme mientras estuvo a mi lado, gracias.*

*Al tutor Osmany y a Carlos Yasmany por el tiempo dedicado y por la ayuda y apoyo que nos dieron.*

*A las muchachitas del apto y grandes amigas, Ivón, Idaliana, Sahily y Malena por soportarnos en los momentos de stress, por ayudarnos cuando lo necesitamos y por estar siempre en momentos buenos y malos junto a mi, de verdad gracias y aunque no lo crean las quiero y las voy a extrañar.*

*A todos mis amigos del grupo viejo, ese piquete que nunca olvidaré y de los que siempre tendré los mejores recuerdos, Jairol, Gustavo, Juan Carlos y Ani, a los de mi grupo no muy viejo, Kenia, Yaidel, Toledo y Yamisel.*

*A toda mi familia que de una forma u otra han ayudado.*

*A todos mis compañeros y a todos los amigos que tuve la suerte de conocer aquí y de los cuales no escribiré los nombres porque son muchos y no quiero que se quede nadie pero ellos sabes que no por eso dejan de ser importantes.*

*A todos los que de una manera u otra me ayudaron, apoyaron y confiaron en mí, a todos Gracias.*

# *Dedicatoria*

## *De Ariana*

*A mi tía Maritza, o no se si decir a mi madre, porque ella ha sido para mí mucho más que eso, ha sido mi amiga, mi madre…, es la persona que nunca dudo de mi, que siempre supo que este momento llegaría y sería el mejor, siempre supo guiarme, ayudarme, apoyarme y lo más importante hizo de mi lo que soy hoy, sabes que esto es para tí y por tí, eres quien realmente se merece todo el esfuerzo de mi trabajo, te lo mereces por tus sacrificios, tus consejos, tu ayuda y todo el amor que me diste y me das.*

*A Juan Andrés, quien siempre me apoyó, que me ayudó siempre que lo necesité, que siempre supo confíar en mí y lo más importante por ser mi buen amigo siempre.*

*A Idolidia no solo por haber sido y ser tan especial conmigo, sino por estar siempre ahí para mí y para mi tía, te queremos…*

**RESUMEN**

En muchos de los sistemas informáticos la generación de reportes es un aspecto fundamental, pues mediante ellos se materializa la gestión de la información. En la actualidad existen en el mundo varios sistemas generadores de reportes, tanto libres como propietarios, pero la mayoría tienen como desventajas que son dependientes de una plataforma específica.

En el presente trabajo se realiza el análisis del subsistema Configuración de Datos, de un generador de reportes, el cual presenta entre sus características la independencia total de la plataforma de desarrollo. Este subsistema será utilizado por los otros subsistemas aportándole todos los datos referentes al reporte. Así mismo se realiza el diseño de este subsistema, Configuración de Datos, aplicando patrones de diseño que le aportan claridad, flexibilidad y robustez.

Para el desarrollo de la investigación se empleó como metodología el Proceso Unificado de Desarrollo (RUP). Los artefactos se generaron usando como lenguaje de modelado el Lenguaje Unificado de Modelado (UML) y auxiliados por el Visual Paradigm como herramienta de Ingeniería de Software Asistida por Computadora (CASE). Para garantizar la calidad de los artefactos generados se aplicaron métricas, que arrojaron resultados positivos.

**PALABRAS CLAVES:** Generador de Reportes, Reportes, Requisitos, Casos de uso, Clases, Análisis, Diseño.

# Tabla de Contenido

# INTRODUCCIÓN

La sociedad actual está comandada por las nuevas tecnologías, donde la informática juega un papel fundamental en todos los ámbitos. El desarrollo de las organizaciones demanda una enorme cantidad de información. Las empresas hoy en día están obligadas a tomar decisiones cada vez más precisas y con mayor rapidez, con el uso de las tecnologías se enfrentan esos problemas y se relacionan, buscando la mejor forma de proporcionar la información necesaria, a fin de tomar mejores decisiones. La información es hoy el recurso clave de la economía, de las organizaciones, del mundo cultural y de la política.

Algunos estudiosos de la ciencia de la computación, prácticamente sin darse cuenta predijeron muchos de los adelantos que hoy se ven en la informática y otros que se verán en un futuro no muy lejano. De una manera u otra el uso de la informática se ha echo indispensable en todas las ramas de la sociedad y por esto en muchos países el uso de las tecnologías de la información se encuentran distribuidas en todo este ámbito social, incluyendo a nuestro país, que a pesar de ser pequeño y no contar con los recursos para ser una potencia informática, lucha por tratar de informatizar las ramas más importantes como la salud, la educación y parte de su economía.

En la actualidad, la mayoría de las Organizaciones o Empresas sus actividades fundamentales son manejadas a través de sistemas informáticos, dichos sistemas son los que realizan lo concerniente a las actividades que cualquier individuo desee informatizar, manteniéndose sobre estos sistemas el control y la gestión de la información.

Un sistema informático utiliza dispositivos programables por medio de computadoras, siendo una síntesis de hardware y software, uno de sus objetivos principales es mostrar los resultados que obtiene en la gestión de la información, con un mejor grado de complejidad, detalle y exactitud, según sea solicitado por el cliente y así mismo que tengan un diseño acorde a sus exigencias. Por esta razón existen en el mundo muchas herramientas que permiten mostrar los resultados obtenidos en la gestión de la información, tanto de repositorios como de bases de datos; estas herramientas son las que permiten la generación de reportes, posibilitando la materialización de la gestión de la información de una manera relativamente sencilla.

Entre las herramientas más importantes en la realización de reportes se pueden encontrar el Active Reports, Jasper Reports y Crystal Reports.

Se están desarrollando muchos proyectos productivos en nuestra universidad, pues la docencia está estrechamente relacionada con la producción, ocupando un papel sumamente

importante, estos proyectos productivos brindan una gran cantidad de salidas en forma de reportes, por lo que un sistema que les brinde soporte a estas generaciones sería de gran importancia.

Existen muchas herramientas no solo aquí en la universidad, sino a nivel mundial, que son utilizadas con el fin de generar reportes, pudiendo ser libres o propietarias. Sin embargo, el uso de muchas de las herramientas utilizadas en estos proyectos para la salida de reportes tienen contradicciones o limitantes que impiden una buena calidad a la hora de generar el reporte, y otros no se auxilian de ningún sistema que viabilice el proceso de generar reportes, esto resulta un poco tedioso a la hora de trabajar con esas herramientas o no tener la herramienta que facilite el trabajo, así como también, resultaría un poco agobiante para el desarrollador. La universidad se encuentra inmersa en transformaciones por lo que quiere migrar hacia software libre, es por ello que resultaría conveniente contar con una herramienta que estandarice y viabilice el proceso de generación de reportes y que a su vez no sea propietaria.

El curso pasado se decidió realizar por parte de algunos estudiantes de la Facultad 3 un generador de reportes que sea capaz de facilitar y mejorar el trabajo en los proyectos productivos de la facultad 3, comenzando en el curso 2007-2008 la elaboración de una tesis que en su conjunto planteó la construcción de esta herramienta y que a su vez llevó a cabo la realización de uno de los módulos en los cuales se decidió dividir el proyecto en el que se va a trabajar. Dándole continuidad a esa tesis, se trabajará en otro módulo que con el ya realizado y otros darán paso a la construcción de esta herramienta.

Para lograr la construcción exitosa de este sistema es necesario que los desarrolladores tengan una entrada correcta de los artefactos, que les permita implementar el sistema con calidad. Obtener esta entrada implica realizar correctamente las actividades correspondientes al flujo de trabajo Análisis y Diseño.

Partiendo de esta situación, se identifica el siguiente **problema**:

¿Cómo obtener artefactos de entrada correctos que permita a los desarrolladores implementar un sistema para la generación de reportes?

De esta forma, se define como **objetivo general** del presente trabajo, realizar el análisis y diseño del subsistema Configuración de Datos de un sistema para la generación de reportes, que permita a los desarrolladores implementarlo correctamente, logrando de esta forma hacer

un aporte y brindar una ayuda a todos los proyectos de la Facultad3 que realicen reportes.

De este objetivo se derivan los siguientes **objetivos específicos**:

- Realizar el estado del arte de los sistemas generadores de reportes, además de las principales herramientas de este tipo que existen en el mundo.
- Obtener los requisitos funcionales y no funcionales del subsistema a trabajar.
- Estructurar el modelo del sistema en función de los requisitos obtenidos.
- Elaborar los artefactos correspondientes al análisis y diseño del módulo Configuración de Datos de un Sistema para la Generación de Reportes.

Se define como **objeto de estudio** todo el Proceso de Generación de Reportes, centrando la investigación en el Análisis y Diseño, identificando el mismo como **campo de acción.**

La **hipótesis** es la siguiente, si se analiza y se diseña correctamente un sistema para la generación de reportes, se podrán obtener artefactos de entrada correctos que permitan a los desarrolladores implementar dicho sistema, facilitando así la estandarización del proceso de generación de reportes.

En vista a dar cumplimiento a los objetivos planteados se definieron las siguientes **tareas de investigación:**

- Asimilación del trabajo al cual se le dará continuidad.
- Actualización del estado del arte.
- Estudio y selección de los patrones de diseño a utilizar en los modelos y en la solución propuesta al problema.
- Realización del análisis y diseño del módulo a trabajar.

Para el desarrollo de las tareas de investigación se emplearon los siguientes **métodos científicos**:

## Métodos Teóricos:

**Analítico – Sintético:** Se utilizó este método para descomponer el tema de los generadores de reportes en sus partes, de modo que permitiera analizar su contenido más profundamente. Además para entender fenómenos relacionados con el proceso de generación de reportes, así

como las tendencias actuales del desarrollo de software, y así de esta forma se arribaron a conclusiones respecto a la investigación.

**Histórico – Lógico:** A través de estos métodos se profundizó en las tendencias, regularidades y cualidades del objeto de estudio y en los argumentos al problema científico de la investigación de forma ordenada históricamente. Permitió constatar teóricamente la evolución del proceso de generación de reportes, así como las herramientas actuales que permiten la materialización de este proceso.

**Inductivo – Deductivo:** A partir del razonamiento y análisis de casos particulares se obtuvieron propuestas a la solución del problema donde se reflejaron los aspectos comunes de los casos analizados. Además de conocer las relaciones entre los distintos elementos de la investigación para poder obtener las generalizaciones y especificaciones del tema.

## Métodos Empíricos:

**Entrevista:** Se realiza para obtener información de las características que se desean para un sistema generador de reportes, así como otros detalles de lo relacionado con la generación de reportes en los proyectos productivos de la Universidad.

### Resultados Esperados:

La obtención de modelos detallados del Módulo Acceso a Datos de un sistema para la generación de reportes dinámicos.

- Especificación de requisitos funcionales y no funcionales del software.
- Modelo de análisis del subsistema Configuración de Datos.
  - Diagrama de clases del análisis
  - Diagramas de colaboración del análisis
- Modelo del diseño del subsistema Configuración de Datos.
  - Diagrama de clases del diseño del subsistema Configuración de Datos
  - Diagramas de secuencia del diseño del subsistema Configuración de Datos

Estructuración del contenido con una breve explicación de sus partes.

El presente trabajo de diploma se ha estructurado de la siguiente forma:

**Capítulo 1:** En este capítulo se hace referencia al proceso de desarrollo de software, haciéndose énfasis en las metodologías de desarrollo de software, tecnologías y tendencias actuales que pueden ser útiles en el desarrollo de la propuesta de solución y centrando un poco más la investigación en el proceso de generación de reportes. En dependencia de la metodología seleccionada se realiza un estado del arte del análisis y diseño, y los principales artefactos que se obtienen como resultado de este, así como de las herramientas que se utilizan para generar estos artefactos y los roles que en este intervienen. Se realiza un estado del arte de las principales herramientas para la generación de reportes que existen en el mundo, un estudio de los patrones y dentro de ellos más específicos los de casos de uso y los de diseño.

**Capítulo 2:** Este capítulo constituye la propuesta de solución para estandarizar el proceso de generación de reportes en la Facultad3. Para describir la solución se realiza el modelo de dominio, un glosario de términos para definir los diversos conceptos que serán utilizados, así como las relaciones que entre estos conceptos se establecen, en aras de lograr la utilización de un vocabulario común. Se realiza la especificación de los requerimientos funcionales y no funcionales que debe presentar la solución a construir. Se determinan los casos de uso del sistema y los actores que interactuarán con éste, elaborándose una descripción en formato expandido de cada uno de los casos de uso, así como el análisis y diseño del sistema con los diagramas correspondientes a este flujo, conjuntamente con esto se exponen los patrones de diseño empleados en la solución.

**Capítulo 3:** Este capítulo constituye el análisis de los resultados, en el mismo se aplican métricas a la especificación de los requisitos, al DCUS y al modelo de diseño, para medir la calidad de cada uno de estos artefactos.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## *1.1 Introducción*

En este capítulo se hace referencia al proceso de desarrollo de software, haciéndose énfasis en las metodologías de desarrollo de software, tecnologías y tendencias actuales que pueden ser útiles en el desarrollo de la propuesta de solución y centrando un poco más la investigación en el proceso de generación de reportes. En dependencia de la metodología seleccionada se realiza un estado del arte del análisis y diseño, y los principales artefactos que se obtienen como resultado de este, así como de las herramientas que se utilizan para generar estos artefactos y los roles que en este intervienen. Se realiza un estado del arte de las principales herramientas para la generación de reportes que existen en el mundo, destacando las más propuestas en el mercado en estos últimos tiempos, se estudiaron los patrones y dentro de ellos más específicos los de casos de uso y los de diseño.

## *1.2 Ingeniería de Software*

En la actualidad en la industria del software hay tendencia al crecimiento y la complejidad de los sistemas que se construyen, esto se debe al hecho de que los computadores son cada vez más potentes y los usuarios por tanto esperan más de ellos. La situación del software actual no es del todo favorable, pues existe un pobre desempeño de las organizaciones de software, dificultades con las mejoras de los procesos de software, además de esto, los proyectos no cumplen con los plazos de tiempo ni de presupuesto establecidos, puesto que se exige mayor calidad y productividad en menos tiempo y hay insuficiente personal calificado; por lo que se puede decir que las fallas de los proyectos de software se deben fundamentalmente a los siguientes factores:

- Planificación irreal.
- Mala calidad del trabajo.
- Personal inadecuado.
- Cambios no controlados.

Para solucionar estos problemas que atentan contra el desarrollo de software de calidad, para desarrollar un software con éxito, que satisfaga las necesidades de los usuarios, que funcione impecablemente durante mucho tiempo, que sea fácil de modificar y fácil de utilizar se necesita disciplina, o sea, un enfoque de ingeniería. Cualquier camino que siga una empresa de software para obtener buena calidad implica que tiene que mejorar el **proceso de desarrollo**

**de software**, por lo tanto, se requiere utilizar los métodos y procedimientos de la Ingeniería y Gestión de software.

Roger Pressman define la Ingeniería del Software como una "disciplina o área de la Informática o Ciencias de la Computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelve problemas de todo tipo. (Pressman, 1998)

El término Ingeniería del Software ha sido definido por otros autores acreditados y organismos internacionales profesionales de prestigio tales como IEEE o ACM:

- Es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software. (Zelkovitz, 1976)
- Es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software. (Bohem, 1976)
- Trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales. (Bauer, 1972)
- La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software. (IEEE, 1993)

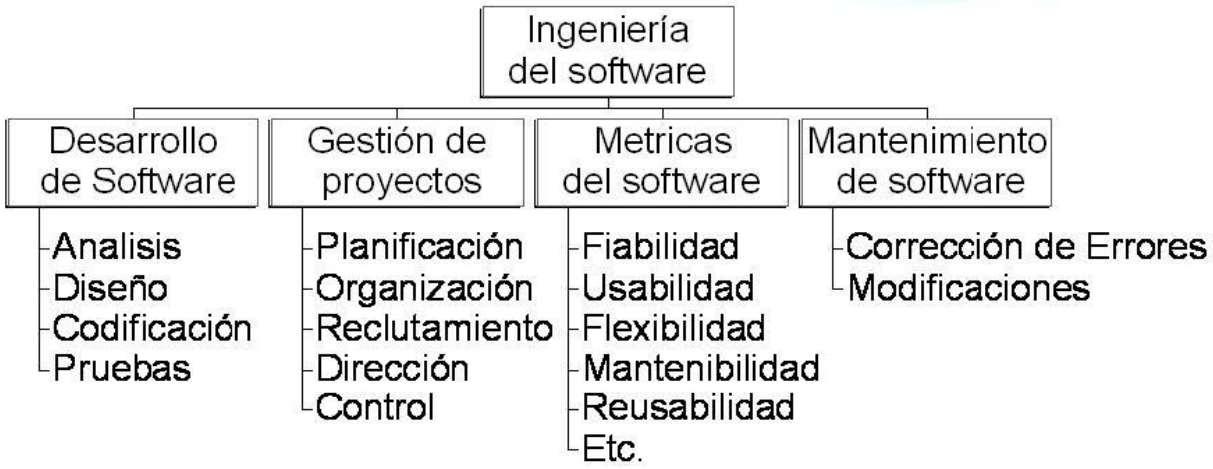



**Figura 1.1.** Modelo de la Ingeniería de Software (Thayer, 1988) (Montesa Andrés)

Después de estudiarse varias definiciones de Ingeniería de Software se puede concluir que las buenas prácticas de la Ingeniería y la calidad de Software permiten crear equipos de alto rendimiento y comprometidos que producen proyectos más exitosos porque están en plazo, en presupuesto, tienen un manejo efectivo de su trabajo y satisfacen las necesidades de los usuarios. Esto se debe a que el equipo de desarrollo trabaja guiado por un proceso, una metodología, un lenguaje y una disciplina común, que comprende todos los aspectos de la producción de software desde las etapas iniciales, hasta el mantenimiento de este después de que se utiliza.

## *1.3 Proceso de Desarrollo del Software*

Un proceso define "quién" está haciendo "qué", "cuándo" y "cómo" para alcanzar un determinado objetivo (Rumbaugh & Grady, 2000)

Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto. (Rumbaugh & Grady, 2000)

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. (Zavala, 2000)

Un proceso de desarrollo de software comprende todas las actividades realizadas por el equipo de desarrollo de software para traducir los requisitos del usuario en un sistema software.

### 1.3.1 Tendencias y metodologías del desarrollo de software

Un aspecto a considerarse hoy día es la exigencia de la industria del software en términos de productividad, calidad y mantenibilidad de los sistemas, lo que lleva a las organizaciones a la necesidad de reducir el esfuerzo en el desarrollo del proyecto, el tiempo en el ciclo de vida y el costo sin afectar la calidad, además de satisfacer los requerimientos y los plazos de entrega exigida por los clientes. (García Ávila, 2007)

Se ha evidenciado la necesidad de la implantación de metodologías de desarrollo con el fin de solucionar la problemática que resulta de la escasa documentación de los sistemas y de la falta de comunicación con los usuarios durante el proceso de desarrollo, lo que genera productos que no responden totalmente a las necesidades de los usuarios. En este sentido, la ingeniería de software define la necesidad de utilizar un conjunto de métodos, procedimientos, técnicas y

herramientas que facilitan la construcción de un sistema informático en el tiempo requerido y con calidad.

(Pressman, 1998) plantea que en el proceso de desarrollo de software lo que ocurre es que:

- No se utilizan eficazmente las metodologías modernas de desarrollo del software ni las herramientas de Ingeniería del Software asistida por computadora (CASE), que son más importantes que el hardware más moderno para conseguir una buena calidad y productividad.
- No existe una adecuada descripción formal y detallada del ámbito de la información, funciones, rendimiento, interfaces, ligaduras de diseño y criterios de validación por una mala comunicación entre clientes y analistas.
- No se garantiza la calidad desde el inicio del proyecto, ni se aplica la revisión técnica formal que es un filtro de calidad muy efectivo para encontrar defectos en el software.

Todos estos problemas se pueden evitar realizando una buena gestión de los proyectos de software, lo que se traduce en la selección de una buena metodología.

No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable.

**Ventajas de tener una buena metodología.**

- Mejora de los procesos de desarrollo.
  - Todas las personas del proyecto trabajan bajo un marco común.
  - Estandarización de conceptos, actividades y nomenclaturas.
  - Actividades de desarrollo apoyadas por procedimientos y guías.
  - Resultados del desarrollo predecibles.
  - Uso de herramientas de ingeniería de software.
  - Planificación de actividades en base a un conjunto de tareas definidas y a la experiencia en otros proyectos.
  - Recopilación de mejores prácticas para proyectos futuros.
- Mejora de los productos.
  - Se asegura que los productos cumplen con los objetivos de calidad propuestos.
  - Detección temprana de errores.
  - Se garantiza la trazabilidad de los productos a lo largo del desarrollo.

- Mejora de las relaciones con el cliente.
  - El cliente percibe el orden en los procesos.
  - Facilita al cliente el seguimiento de la evolución del proyecto.
  - Se establecen mecanismos para asegurar que los productos desarrollados cumplen con las expectativas del cliente. (López Barrio, 2005)

**Características deseables de una metodología.**

- Existencia de reglas predefinidas.
- Cobertura total del ciclo de desarrollo.
- Verificaciones intermedias.
- Planificación y control.
- Comunicación efectiva.
- Utilización sobre un abanico amplio de proyectos.
- Fácil formación.
- Herramientas CASE.
- Actividades que mejoren el proceso de desarrollo.
- Soporte al mantenimiento.
- Soporte de la reutilización de software. (García Rubio & Bravo Santos, 2007)

Existen en la actualidad múltiples metodologías, estas pueden dividirse en dos categorías: **Tradicionales y Ágiles**. Se realizará un estudio de las metodologías que se usan en el desarrollo de software, y luego de tener un conocimiento previo se realizará la selección de la metodología que guiará el proceso de desarrollo del presente trabajo.

### 1.3.1.1 Metodologías Tradicionales

Las metodologías tradicionales se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, los roles, las herramientas y notaciones que se usarán, incluyendo modelado y documentación detallada. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Estas metodologías se caracterizan por ser rígidas y dirigidas por la documentación que se genera en cada una de las actividades desarrolladas. Este esquema "tradicional" para abordar el desarrollo de software es efectivo en proyectos de gran envergadura donde por lo general la

gestión es el principal problema, por lo que se necesita un proceso gestionado de forma estricta.

**Ejemplos de metodologías tradicionales:**

**SPICE:** metodología de desarrollo en la que se definen un conjunto de buenas prácticas para la mejora gradual de los procesos de ingeniería.

**Rational Unified Process (RUP):** define un ciclo de vida iterativo priorizando el uso de lenguajes de modelado, casos de uso y centrado en la arquitectura.

**Microsoft Solution Framework (MSF):** metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. (López Barrio, 2005)

### 1.3.1.2 Metodologías Ágiles

Las metodologías ágiles están especialmente orientadas para proyectos pequeños. Constituyen una solución con una elevada simplificación, a pesar de ello no renuncian a las prácticas esenciales para asegurar la calidad del producto. Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, nuevas tecnologías, etc.

**El Manifiesto Ágil es un documento que resume la filosofía ágil. Los principios de este manifiesto se muestran en el Anexo 1**

**Ejemplos de Metodologías Ágiles:**

- XP o Programación Extrema
- Scrum
- Crystal
- FDD o Desarrollo Manejado por Funcionalidades. (Canós, Letelier, & Penadés)

**Comparación entre las metodologías ágiles y las metodologías tradicionales Anexos Tabla 1**

### 1.3.2 RUP

El Proceso Unificado de Desarrollo (RUP) es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. La versión que se ha estandarizado vio la luz en 1998 y se conoció en sus inicios como Proceso Unificado de Rational 5.0; de ahí las siglas con las que se identifica a este proceso de desarrollo.

**Como RUP es un proceso, en su modelación define como sus principales elementos:**

- Trabajadores (“quien”)
- Actividades (“como”)
- Artefactos (“que”)
- Flujo de Actividades (“cuando”)

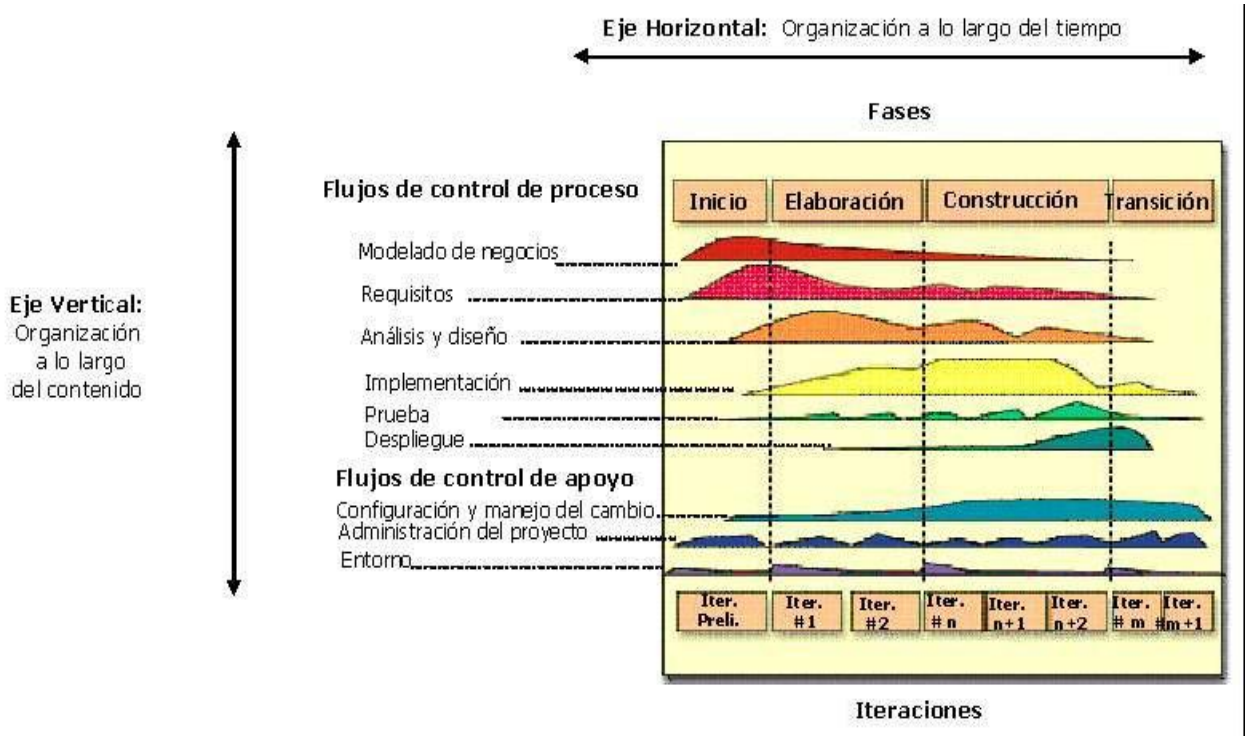



**Figura 1.2.** Flujos de trabajo y fases de la metodología RUP (Teleformación)

Como ilustra la Figura 1.2 el Proceso Unificado de Desarrollo (RUP-Rational Unified Process) es un proceso pensado en dos dimensiones. El mismo se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los clientes. Cada ciclo consta de cuatro fases. Cada fase termina con un hito. (Jacobson,

Ivar, Rumbaugh & Booch, 2004 pág. 8 y 10) **Las cuatro fases son:**

- **Conceptualización (Concepción o Inicio):** Se desarrolla una descripción del producto final y se presenta el análisis del negocio para el producto. Se identifican y priorizan los riesgos más importantes, se estima el proyecto de manera aproximada.
- **Elaboración:** Se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. Se realizan los casos de uso más críticos que se identificaron en la fase de inicio. El resultado de esta fase es la línea base de la arquitectura.
- **Construcción:** Durante esta fase se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo. La descripción evoluciona hasta convertirse en un producto preparado para ser entregado a la comunidad de usuarios. Al final de esta fase el producto contiene todos los casos de uso que la dirección y el cliente han acordado para el desarrollo de esta versión.
- **Transición:** Cubre el período durante el cual el producto se convierte en versión sea, un número reducido de usuarios con experiencia prueba el producto e defectos y deficiencias. Esta fase conlleva a actividades como la fabricación, del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección que se encuentren tras la entrega. (Rumbaugh & Grady, 2000)

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. En la Figura 1.2: RUP en Dos Dimensiones se representa el proceso en el que se grafican los flujos de trabajo y las fases y muestra la dinámica expresada en iteraciones y puntos de control.

Flujos de trabajo de RUP son:

- **Modelamiento del negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.

- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.
- **Instalación:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- **Administración del proyecto:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- **Administración de configuración y cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

El Proceso Unificado está basado en componentes. Utiliza el nuevo estándar de modelado visual, el Lenguaje Unificado de Modelado (UML), y se sostiene sobre tres ideas básicas, casos de uso, arquitectura, y desarrollo iterativo e incremental. Esto es lo que hace único al Proceso Unificado. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

Por lo tanto, RUP es un proceso dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.

**Un proceso dirigido por casos de uso. ¿Qué significa esto?**

Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema. (Kruchten, 2000)

Los casos de uso no son solo una herramienta para especificar los requisitos de un sistema. También guían su diseño, implementación y prueba; guían el proceso de desarrollo. Basándose en el modelo de casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso. Los desarrolladores revisan cada uno de los sucesivos modelos para que sean conformes al modelo de casos de uso. Los ingenieros de prueba prueban la implementación para garantizar que los componentes del modelo de

implementación implementan correctamente los casos de uso. De este modo los casos de uso

no solo inician el proceso de desarrollo, sino que les proporcionan un hilo conductor. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

La **arquitectura** de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo. (Kruchten, 2000). Podemos afirmar que el Proceso Unificado está centrado en la arquitectura, y esto se debe fundamentalmente a que los casos de uso solamente no son suficientes. Se necesitan más cosas para conseguir un sistema de trabajo. Cada producto tiene tanto una función como una forma. Ninguna es suficiente por sí misma. Estas dos fuerzas deben equilibrarse para obtener un producto con éxito. En esta situación, la función corresponde a los casos de uso y la forma a la arquitectura. La arquitectura nos da una clara perspectiva del sistema completo, necesaria para controlar el desarrollo. Se necesita una arquitectura para: comprender el sistema, organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema. El desarrollo de un producto software comercial supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o mini-proyectos.

**¿Por qué un desarrollo iterativo e incremental?**

Para obtener un software mejor. Para cumplir los hitos principales y secundarios con los cuales se controla el desarrollo.

Los conceptos - dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental- son de igual importancia. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración. La eliminación de una de las tres ideas reduciría drásticamente el valor del Proceso Unificado. Es como un taburete de tres patas. Sin una de ellas el taburete se cae. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

RUP identifica seis de las llamadas **mejores prácticas** con las que define una forma efectiva de trabajar para los equipos de desarrollo de software. Ellas son:

- **Desarrollo de software iterativo:** Para los sofisticados sistemas de software que se desarrollan en la actualidad es necesario un enfoque iterativo, que permita una mayor comprensión del problema a través de sucesivos refinamientos, y lograr una solución

eficaz a través de múltiples iteraciones. Este enfoque iterativo del desarrollo propicia que en cada etapa del ciclo de desarrollo se aborden los temas de mayor riesgo, reduciendo significativamente el perfil de riesgo del proyecto. (Kruchten, 2000)

- **Administrar requerimientos:** El Proceso Unificado describe como elicitar, organizar, documentar y seguir los cambios de las funcionalidades y restricciones requeridas. Las nociones de caso de uso y escenario han demostrado ser una excelente vía de capturar los requerimientos funcionales y de asegurarnos que estos facilitan el diseño, la implementación y las pruebas; lo que hace más probable que el sistema final satisfaga las necesidades de los usuarios finales. (Jacobson, 2004)

- **Desarrollo basado en componentes:** El Proceso Unificado apoya el desarrollo de software basado en componentes. Los componentes son módulos no triviales, subsistemas que cumplen una clara función. El Proceso Unificado proporciona un enfoque sistemático para la definición de la arquitectura utilizando componentes tanto nuevos como existentes, estos son unidos en una arquitectura bien definida. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes. (Brown, 1996)

- **Modelado Visual:** Las abstracciones visuales ayudan a comunicar los diferentes aspectos del software, ver cómo los elementos del sistema se comunican entre sí, asegurarse que los componentes están acordes con el código, mantener la coherencia entre el diseño y la implementación, y promover la comunicación inequívoca. (Jacobson, Ivar, Rumbaugh, Booch, & Booch, 2004)

- **Verificación continua de la calidad:** Poco rendimiento de las aplicaciones y poca fiabilidad son algunos de los factores que inhiben la aceptación de las aplicaciones de software actuales. Por lo tanto, la calidad debe revisarse con respecto a los requisitos sobre la base de la fiabilidad, funcionalidad, el rendimiento de la aplicación y el rendimiento del sistema. La evaluación de la calidad se construye en el proceso, en todas las actividades, con la participación de todos los miembros del equipo, utilizando criterios y mediciones objetivas, y que no sean tratados como una ocurrencia tardía o una actividad realizada por un grupo separado.

- **Gestión de los cambios:** Poder realizar el seguimiento de los cambios es esencial en un entorno en el que el cambio es inevitable. El Proceso Unificado de Desarrollo describe como controlar, seguir y supervisar los cambios para lograr un desarrollo iterativo exitoso. Proporciona una guía para establecer espacios de trabajo seguros

para cada desarrollador, proporcionando aislamiento de los cambios realizados en otros lugares de trabajo y controlando los cambios en todos los artefactos de software.

### 1.3.3 XP

La Programación Extrema es una metodología ligera de desarrollo de software, que surge como respuesta y posible solución a los problemas derivados del cambio en los requerimientos. En la mayoría de los casos se plantea como una metodología a emplear en los proyectos con riesgos de requisitos muy cambiantes. Se plantea que «Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que se sabe que el cambio va a suceder; el problema es la incapacidad de adaptarse a dicho cambio cuando éste tiene lugar» (Escribano, 2002). XP es una metodología orientada al cliente y de iteraciones cortas.

**Lo fundamental de XP es:**

La comunicación: Entre los usuarios y los desarrolladores.

La simplicidad: Al desarrollar y codificar los módulos del sistema.

La retroalimentación: Concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales. (Becerra González & Durand Martínez, 2007-2008)

**Fases de la metodología XP:**

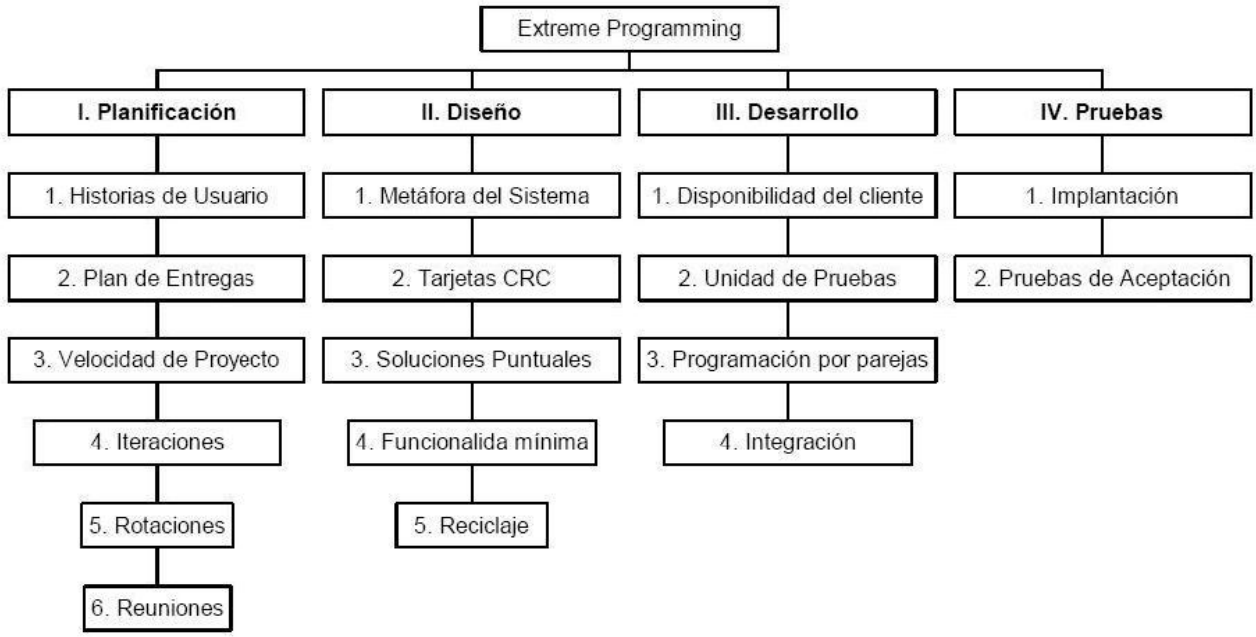



**Figura 1.3.** Fases de la metodología XP (Gerardo Fernándes, 2002)

XP es un proceso muy orientado a la implementación, en el que se genera poca documentación y en que la funcionalidad exacta del sistema final no se define nunca formal y contractualmente. Es por eso que este método es más aplicable para desarrollos internos. (Sánchez, Maglema, & Vázquez Baños, 2007)

### 1.3.4 Justificación de la metodología a utilizar.

Después del estudio de ambas metodologías se decidió seleccionar la metodología RUP para guiar el proceso de desarrollo, pues a pesar de ser una metodología tradicional, o sea, rígida y dirigida por la documentación que se genera en cada una de las actividades desarrolladas, es una metodología muy exitosa y utilizada actualmente en proyectos de gran envergadura, además de que posee muchas características que ayudaron a su selección, entre ellas se pueden citar:

- Establece rigurosamente las actividades involucradas, los roles, las herramientas y notaciones que se usarán, incluyendo modelado y documentación detallada, así como los artefactos que se deben producir, esto último la convierte en una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.
- Este proceso tiene tres características que lo hacen único: Dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.

- Contempla seis de las llamadas buenas prácticas para el desarrollo de un software.
- Aumenta la productividad del equipo, pues proporciona un acceso común a todos los miembros del proyecto al mismo conocimiento base, por lo que sin importar en que fase del Proceso Unificado se encuentre algún miembro trabajando todos comparten el mismo lenguaje, proceso y visión de cómo desarrollar software.
- Se emplea fundamentalmente en proyectos que se desarrollarán a largo plazo.
- Es una metodología orientada al proceso.

Como el objetivo general de esta tesis es realizar el análisis y el diseño de un subsistema de un generador de reportes, se decidió seleccionarla en lugar de XP, pues este último es un proceso muy orientado a la implementación, por lo que para lograr un análisis y diseño exitoso RUP es la más adecuada.

Con respecto a la metodología RUP el Dr. Xavier Ferré Grau en su tesis de doctorado (Ferrer Grau, 2005) afirma que el proceso que está tomando mayor atención en la Ingeniería de Software en la actualidad es el Proceso Unificado y que esto es debido a que sus impulsores son los mejores metodólogos del desarrollo orientado a objetos de la década del 90, James Rumbaugh, Ivar Jacobson y Grady Booch. Sostiene además que adopta un verdadero enfoque iterativo y que es el más aplicado en proyectos reales. Alega que Rational Unified Process (RUP) es una particularización del modelo de proceso representado por el Proceso Unificado.

Junto al Lenguaje Unificado de Modelado (UML) constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. (Tejera Hernández & Sánchez Echevarría, 2007)

## *1.4 El Lenguaje Unificado de Modelado*

El modelado visual es el proceso de tomar la información de un modelo y representarla gráficamente utilizando un conjunto de elementos gráficos estándares. El modelado visual facilita la comunicación entre usuarios, desarrolladores, analistas, probadores, gerentes y todos los participantes en el proyecto. (Boggs & Boggs, 2002)

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está

pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. (Jacobson, Ivar, Rumbaugh & Booch, 2004) El UML cumple con la función de servir de enlace entre quien tiene la idea del sistema y el desarrollador, ya que le ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien está involucrado en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo.

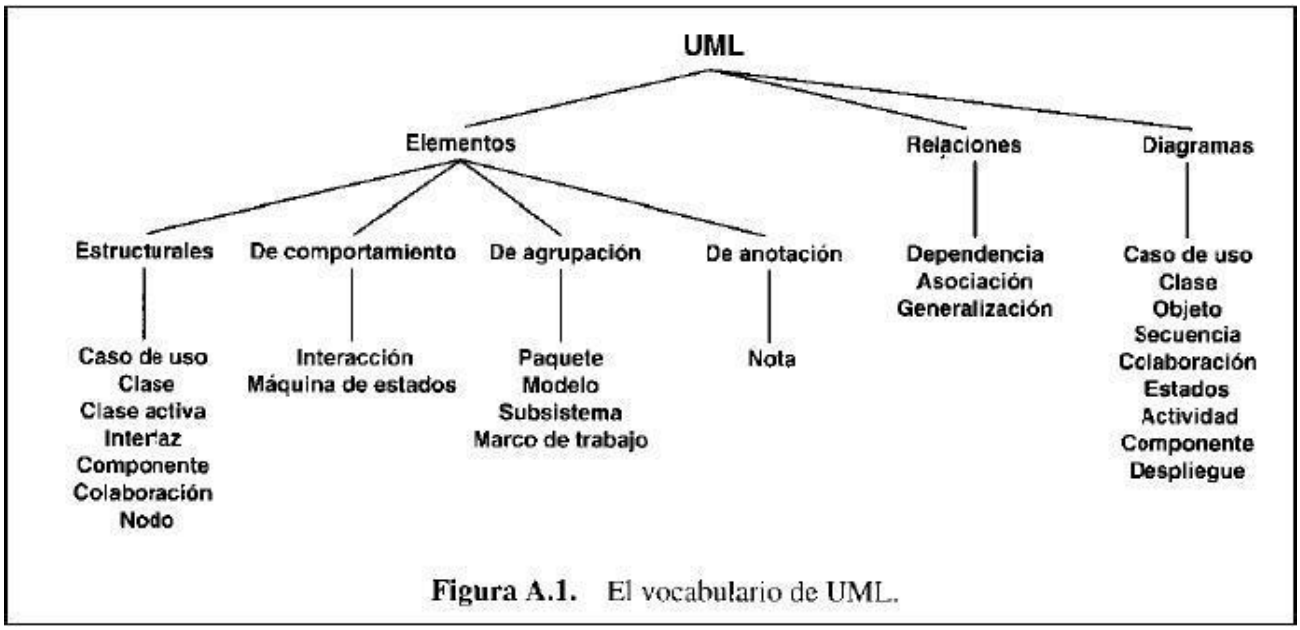



**Figura 1.4. El vocabulario de UML**

UML proporciona una organización en el proceso de diseño de forma tal que analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. (Schmuller, 2000). El UML es un lenguaje para construir modelos; no guía al desarrollador en la forma de realizar análisis y diseño orientado a objetos, ni le indica cuál proceso de desarrollo adoptar. Los autores del lenguaje UML -Booch, Jacobson y Rumbaugh- hicieron un excelente servicio a la comunidad de la tecnología orientada a objetos al crear un lenguaje estandarizado de modelado que es elegante, expresivo y flexible.

Prescindiendo de la aceptación que pueda tener, este lenguaje recibió la aprobación de facto en la industria, pues sus creadores representan métodos muy difundidos de la primera generación del análisis y diseño orientado a objeto. Muchas organizaciones dedicadas al desarrollo de software y los proveedores de herramientas de CASE (Computer Aided Software Engineering) lo adoptaron, y muy probablemente se convierta en el estándar mundial que utilizarán los desarrolladores de herramientas CASE. (Larman, 1999) Algunas de las

propiedades de UML como lenguaje de modelado estándar son:

- Concurrencia: es un lenguaje distribuido y adecuado a las necesidades de conectividad actual y futura.
- Ampliamente utilizado por la industria desde su adopción por OMG5.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clases, componentes y nodos.
- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.
- Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones, que sirven para evaluar el estado de las máquinas. (Joyanes Aguilar, 2000)

Se decidió utilizar UML pues la metodología seleccionada (RUP) lo emplea de manera eficiente, y de esta forma permite evolucionar adecuadamente en el diseño e implementación de un sistema informático.

## *1.5 Herramienta CASE*

El incremento del desarrollo de software hizo que se creara un soporte automatizado para el desarrollo y mantenimiento de software. Este es el llamado “ingeniería del software asistida por computadora”.

Una de las razones para la creación de las herramientas CASE fue el incremento en la velocidad de desarrollo de los sistemas, permitiendo a los analistas tener más tiempo para el análisis y diseño, y minimizar el tiempo para codificar y probar. Una herramienta CASE es un producto computacional enfocado a apoyar una o más técnicas dentro de un método de desarrollo de software. (Jarzabek & Huang, 1998) De acuerdo con Kendall y Kendall (Kendall & Kendall, 1997) la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo. Su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Las herramientas CASE son usadas en algunas de las fases de desarrollo de sistemas de información, incluyendo análisis, diseño y programación. El objetivo fundamental de las herramientas CASE es proveer un lenguaje para describir el sistema general que sea

suficientemente explícito para generar todos los programas necesarios (Electronic Computer Glossary, 1995).Son muchos los beneficios que pueden proveer las herramientas CASE en todas las etapas del proceso de desarrollo de software, algunas de ellas son:

- Hacer el trabajo de diseño de software más fácil y agradable.
- Verificar el uso de todos los elementos en el sistema diseñado.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en las bases de datos.
- Generar estructuras de código.
- Reducción del costo de producción de software. (Kendall & Kendall, 1997)

El uso de las herramientas CASE permite una mejora en la calidad de los desarrollos realizados, pero para lograr esto además de la propia herramienta CASE es necesario contar con una organización y una metodología de trabajo. Estas herramientas permiten la reutilización del código, la portabilidad del software y la estandarización de la documentación.

### 1.5.1 Rational Rose Enterprise Edition 2003

**Rational Rose:** Esta herramienta que domina en el mercado mundial, fue desarrollada por los creadores de UML y emplea este lenguaje para modelar. Es una de las más potentes desarrolladas hasta el momento, pero consume muchos recursos, lo que hace que si no se presenta la tecnología adecuada, resulte incómodo trabajar con ella.

Rational Rose Enterprise Edition es una herramienta CASE que soporta el modelado visual con UML, ofreciendo distintas perspectivas del sistema. Da soporte al Proceso Unificado de Rational (RUP). Algunas de sus características son:

- Diseño dirigido por modelos que redundan en una mayor productividad de los desarrolladores.
- Diseño centrado en casos de uso y enfocado al negocio, que generan un software de mayor calidad.
- Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables.
- Solo permite trabajar sobre el sistema operativo Windows.

- Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.
- Desarrollo Iterativo.
- Generador de Código.
- Ingeniería Inversa.

### 1.5.2 Visual Paradigm for UML

**Visual Paradigm:** Esta herramienta es multiplataforma y está diseñada para desarrollar software con programación orientada a objetos. Es una herramienta de desarrollo que se integra al IDE de Eclipse. Su licencia cuesta más de 2300 dólares y además, es importante destacar que posee librerías privadas, lo que le imposibilita a los programadores utilizar sus propias librerías para generar código de acceso a datos.

Visual Paradigm 6.0 es una herramienta CASE orientada a UML, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Visual Paradigm genera toda la documentación de lo que se hace cumpliendo con estándares establecidos. Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así como obtener diagramas a partir de código. Esta es precisamente una gran ventaja puesto que el sistema será desarrollado en .Net. El análisis textual es una técnica útil y práctica para la captura de los requisitos del sistema y la identificación de las clases candidatas, Visual Paradigm es una de las pocas herramientas CASE que soporta el análisis textual. Tiene disponibilidad en múltiples plataformas y en múltiples versiones. Esta característica es muy importante pues por ejemplo el Rational Rose, que es una herramienta muy recomendada y además profesional, tiene una desventaja en su contra pues obliga al usuario a desarrollar en máquinas con el sistema operativo Windows, mientras que el Visual Paradigm está disponible para varios sistemas operativos como Windows, Linux, Unix. Se decidió seleccionar esta herramienta CASE fundamentalmente por esta característica.

## *Ingeniería de Requisitos*

La mejor forma de medir la calidad de un sistema es por el grado de satisfacción que tenga el cliente. Lograr un entendimiento común y efectivo entre los usuarios y el equipo de proyecto, así como también entre los miembros de este último, con el objetivo de llegar a un entendimiento de lo que hay que hacer, es la clave del éxito en la producción de software.

Durante muchos años las aplicaciones han fallado (no se culminaron o no se usaron) porque existieron incongruencias entre lo que el usuario quería, lo que realmente necesitaba, lo que interpretaba cada miembro del proyecto y lo que realmente se obtiene. Esto se resume a que existen dificultades en uno de los flujos de trabajo más importantes del ciclo de vida del software: el Levantamiento de Requisitos, un mal trabajo durante este flujo de trabajo afecta de forma negativa al sistema final si se realizara. Ninguna otra parte es más difícil de rectificar después.

La ingeniería de requisitos consiste en un conjunto de actividades y transformaciones que pretenden comprender las necesidades de un sistema software y convertir la declaración de estas necesidades en una descripción completa, precisa y documentada de los requerimientos del sistema, siguiendo un determinado estándar. (Marqués, 2005)

### Técnicas empleadas en la captura de requisitos

La captura de requisitos es la actividad a través de la cual el equipo de desarrollo de un sistema de software obtiene, de cualquier fuente de información disponible, las necesidades que debe cumplir dicho sistema. Por la complejidad que todo este proceso puede implicar, la ingeniería de requisitos define técnicas que permiten hacer este proceso de una forma más eficiente y precisa. Estas técnicas son:

- Entrevistas
- JAD6
- Tormenta de ideas
- Casos de Uso
- Cuestionarios
- Comparación de Terminología

### Entrevistas

Resulta una técnica muy aceptada y de un uso ampliamente extendido dentro de la ingeniería de requisitos. Las entrevistas le permiten al analista obtener conocimiento del problema y comprender los objetivos de la solución esperada. A través de esta técnica el equipo de trabajo

se acerca al problema de una forma natural. Existen muchos tipos de entrevistas. Básicamente, la estructura de la entrevista abarca tres pasos: identificación de los entrevistados, preparación de la entrevista, realización de la entrevista y documentación de los resultados.

La entrevista no es una técnica sencilla de aplicar, requiere que el entrevistador sea experimentado y tenga capacidad para elegir bien a los entrevistados y obtener de ellos toda la información posible en un período de tiempo siempre limitado. Es por ello que la preparación de la entrevista juega un papel fundamental. (José Escalona & Koch, 2002)

### JAD

Esta técnica se realiza como una alternativa a las entrevistas. Es una práctica de grupo que se desarrolla durante varios días y en la que participan analistas, usuarios, administradores del sistema y clientes. Está basada en cuatro principios fundamentales: dinámica de grupo, el uso de ayudas visuales para mejorar la comunicación, mantener un proceso organizado y racional y una filosofía de documentación WYSIWYG (What You See Is What You Get, lo que ve es lo que obtiene), es decir, durante la aplicación de la técnica se trabajará sobre lo que se generará. En cada una de las sesiones que se realizan se establecen los requisitos de alto nivel a trabajar, el ámbito del problema y la documentación, llegándose a una serie de conclusiones que se documentan. En cada sesión se van concretando más las necesidades del sistema.

A la hora de ver las ventajas que presenta esta técnica frente a las entrevistas que es la que por tradición más se utiliza, se puede ver que se ahorra tiempo al evitar que a los clientes se les tenga que contrastar por separados, sin embargo se requiere de un grupo de participantes bien integrados y organizados.

### Tormenta de ideas

Es también una técnica de reuniones en grupo, cuyo objetivo es que los participantes muestren sus ideas de forma libre. Esto no es más que la acumulación de ideas y/o información sin evaluar las mismas. Como técnica de captura de requisitos es sencilla de usar y de aplicar, contrariamente al JAD, puesto que no requiere tanto trabajo en grupo como este. Tiene como desventaja que no sirve para obtener de ellos detalles concretos ya que estos se suelen usar solo en los primeros encuentros, a pesar de ofrecer una visión general de las necesidades del sistema.

### Casos de uso

Esta técnica aunque inicialmente se desarrollaron como técnica para la definición de requisitos algunos autores proponen casos de uso como técnica para la captura de requisitos. Los casos de uso permiten mostrar el contorno (actores) y el alcance (requisitos funcionales expresados como casos de uso) de un sistema. Un caso de uso describe la secuencia de interacciones que se producen entre el sistema y los actores del mismo para realizar una determinada función.

La ventaja esencial de los casos de uso es que resultan muy fáciles de entender para el usuario o cliente, sin embargo, carecen de la precisión necesaria si no se acompañan con una información textual o detallada con otra técnica, como pueden ser los diagramas de actividades. (José Escalona & Koch, 2002)

### Cuestionarios

Esta técnica requiere que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario (Checklist). Este cuestionario será cumplimentado por el grupo de personas entrevistadas o simplemente para recoger información en forma independiente de una entrevista. (José Escalona & Koch, 2002)

### Comparación de Terminología

Uno de los problemas que surge durante la elicitación de requisitos es que usuarios y expertos no llegan a entenderse debido a problemas de terminología. Esta técnica es utilizada en forma complementaria a otras técnicas para obtener un consenso respecto de la terminología a ser usada en el proyecto de desarrollo. Para ello es necesario identificar el uso de términos diferentes para los mismos conceptos (correspondencia), misma terminología para diferentes conceptos (conflictos) o cuando no haya concordancia exacta ni en el vocabulario ni en los conceptos (contraste). (José Escalona, y otros, 2002) Las técnicas descritas anteriormente representan las más utilizadas, pero existen otras como el análisis de otros sistemas y el estudio de la documentación que también pueden ser aplicadas para el desarrollo del presente trabajo de diploma.

## *1.6 Patrones*

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones.

Cada patrón describe un problema que ocurre una y otra vez en el ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que se puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces, es un modelo que se puede seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Capturan la experiencia existente y probada para promover buenas prácticas.

De manera más simple, un **patrón** es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

**Existen diversas clases de patrones:**

- De casos de uso
- De análisis
- De arquitectura (divididos en progresivamente en estructurales, sistemas distribuidos, sistemas interactivos, sistemas adaptables)
- De diseño (conductuales, creacionales, estructurales)
- De organización o proceso
- De programación

### 1.6.1 Patrones de Casos de Uso.

Un patrón de caso de uso es un diseño generalmente probado en un modelo de casos de uso, junto a una descripción del contexto en el cual será usado y qué consecuencias tendrá su aplicación en el modelo.

La experiencia en la utilización de casos de uso ha evolucionado en un conjunto de patrones que permiten con más precisión reflejar los requisitos reales, haciendo más fácil el trabajo con los sistemas, y mucho más simple su mantenimiento. Dado un contexto y un problema a resolver, estas técnicas han mostrado ser la solución adoptada en la comunidad del desarrollo de software. Estos patrones se enfocan hacia el diseño y las técnicas utilizadas en modelos de alta calidad, y no en cómo modelar usos específicos.

**Algunos patrones de casos de uso son los siguientes:**

- Reglas de negocio
- Concordancia (Commonality)

- Componente jerárquico (Component hierarchy)
- CRUD (Creating, Reading, Updating, Deleting)
- Sistema de Capas
- Múltiples actores
- Secuencia de casos de uso.

De estos mencionados sólo se abordarán cinco patrones: Reglas de negocio, Concordancia, CRUD, Sistema de capas y Múltiples actores

### Reglas de Negocio

Se basan en la extracción de información originada de las políticas, reglas y regulaciones del negocio de la descripción del flujo y describe la información como una colección de reglas del negocio referenciadas a partir de las descripciones de los casos de uso.

Este patrón es aplicado a todos los casos de uso modelando los servicios que son afectados por las reglas del negocio definidas en la organización. Sin embargo, este patrón no influye en la estructura del modelo de casos de uso. Las reglas son descritas en un documento separado, referenciadas por las descripciones de los casos de usos relevantes. Este patrón es apropiado utilizarlo cuando no hay necesidad de cambiar dinámicamente las reglas del negocio mientras el sistema se este utilizando.

### Concordancia (Commonality)

Extrae una subsecuencia de acciones que aparecen en diferentes lugares del flujo de casos de uso y es expresado por separado.

Consta de 3 casos de uso. El primero llamado subsecuencia común, modela una secuencia de acciones que aparecerán en múltiples casos de uso en el modelo. Los otros casos de uso modelan el uso del sistema que comparte la subsecuencia común de acciones. De manera que deben existir al menos dos de ellos.

### CRUD (Creating, Reading, Updating, Deleting)

CRUD: Completo

**Descripción:** Consiste en un caso de uso llamado Información CRUD o Administrar Información, modelando todas las operaciones que pueden ser realizadas en un segmento de

información, como es: crear, leer, actualizar y eliminar.

**Aplicación:** Este patrón puede ser utilizado cuando todos los flujos contribuyen al cumplimiento del mismo objetivo y son todos cortos y simples.

**Tipo:** Patrón de estructura.

- Extensión concreta o inclusión: Inclusión

**Descripción:** En este patrón hay una relación de inclusión del caso de uso base al caso de uso incluido, el cual puede ser instanciado por si mismo.

**Aplicación:** Este patrón es aplicable cuando un flujo puede ser incluido en flujo de otro caso de uso.

**Tipo:** Patrón de estructura

- Servicio Opcional: Adición

**Descripción:** Contiene dos casos de uso y una relación de extensión. El primer caso de uso modela un uso que es obligatorio en el sistema. El segundo caso de uso modela una adición al primero que puede ser añadida al sistema. Como la parte adicional solo esta expresada en el segundo caso de uso, este es abstracto, esto significa que este no será ejecutado por sí mismo.

**Aplicación:** Este patrón es preferido cuando la parte opcional es una pura adición del caso de uso obligatorio.

**Tipo:** Patrón de estructura

- Cohesión: Rehúso Interno

**Descripción:** Si la subsecuencia de acciones es usada en múltiples lugares en un solo caso de uso, este no necesita extraer la subsecuencia dentro de un caso de uso separado. Esto podría ser descrito separado en una sub-sección en la descripción del caso de uso.

**Aplicación:** Este patrón es preferible usarlo cuando la sub-secuencia común se muestre en varios escenarios dentro de un caso de uso.

**Tipo:** Patrón descriptivo

- Escenario más Fragmentos

**Problema:** El lector debe ser capaz de fácilmente seguir el camino a través del flujo específico o historia en que está interesado, en caso contrario probablemente se frustrará.

**Solución:** Escribir los eventos del flujo principal como un escenario simple sin considerar posibles fallos. Debajo ubicar los fragmentos del flujo que muestran que condición alternativa puede ocurrir.

**Tipo:** Patrón Descriptivo

- Alternativas Exhaustivas, Íntegras

**Problema:** Un caso de uso puede tener muchas alternativas. Faltando algunos recursos los desarrolladores pueden entender mal el comportamiento del sistema y entonces el sistema sería deficiente. Un buen caso de uso describe todas las alternativas importantes a fin de que los diseñadores puedan correctamente localizar problemas potenciales protegiendo a los usuarios de sorpresas desagradables.

**Solución:** Capturar todos los fallos y alternativas que deben ser manejados en el caso de uso. Una vez identificados todos los casos de uso y su flujo principal identificar tantas variaciones como sea posible en el flujo principal. Capturar de forma selectiva todas las variaciones que se desea que el sistema maneje.

**Tipo:** Patrón Descriptivo
(Övergard, y otros, 2004)

### Múltiples actores

#### Roles diferente

Captura la concordancia entre actores manteniendo roles separados. Consiste de un caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso, o sea, interactúan de forma diferente con el caso de uso.

#### Roles comunes

Puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el

punto de vista del caso de uso, solo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

**Sistema de capas**

Estructura el modelo de casos de uso de manera tal que cada caso de uso es definido dentro de una capa, y utiliza relaciones entre los casos de uso en diferentes capas que permitan instanciar los casos de uso para expandir múltiples capas.

En cada uno de los patrones del sistema de capas, existen dos paquetes modelando dos capas, un paquete importa las relaciones del paquete que representa las capas superiores para el paquete que representa las capas inferiores. La relación implica que el contenido (público) del paquete de la capa inferior se torne disponible dentro del paquete de la capa superior. Todas las relaciones entre los elementos determinados en diferentes paquetes de capas, son definidas dentro del paquete de capas superior.

### 1.6.2 Patrones de Diseño

Los patrones de diseño (*design patterns*) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño nombra, abstrae e identifica los aspectos claves de un diseño estructurado común, que lo hace útil para la creación de diseños orientados a objetos reutilizables.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

**Según el libro GOF existen 3 tipos de patrones de diseño:**

- **De Creación:** abstraen el proceso de creación de instancias.
- **Estructurales:** se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- **De Comportamiento:** están relacionados con algoritmos y asignación de responsabilidades entre objetos.

### Relación de principales patrones GoF

### Patrones creacionales

- **Abstract Factory (Fábrica abstracta):** Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- **Builder (Constructor virtual):** Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- **Factory Method (Método de fabricación):** Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- **Prototype (Prototipo):** Crea nuevos objetos clonándolos de una instancia ya existente.
- **Singleton (Instancia única):** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

### Patrones Estructurales

- **Adapter (Adaptador):** Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- **Bridge (Puente):** Desacopla una abstracción de su implementación.
- **Composite (Objeto compuesto):** Permite tratar objetos compuestos como si de uno simple se tratase.
- **Decorator (Envoltorio):** Añade funcionalidad a una clase dinámicamente.
- **Facade (Fachada):** Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- **Flyweight (Peso ligero):** Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
- **Proxy:** Mantiene un representante de un objeto.

### Patrones de Comportamiento

- **Chain of Responsability (Cadena de responsabilidad):** Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- **Command (Orden):** Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- **Interpreter (Intérprete):** Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- **Iterator (Iterador):** Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- **Mediator (Mediador):** Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- **Memento (Recuerdo):** Permite volver a estados anteriores del sistema.
- **Observer (Observador):** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- **State (Estado):** Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- **Strategy (Estrategia):** Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- **Template Method (Método plantilla):** Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- **Visitor (Visitante):** Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

### Patrones de Sistema

- **MVC (Modelo Vista Controlador):** Divide un componente o un subsistema en tres partes lógicas: modelo, vista y controlador, facilitando la modificación o personalización de cada parte.
- **Session (Sesión):** Ofrece una forma de que los servidores de sistemas distribuidos sean capaces de distinguir los clientes, permitiendo que las aplicaciones asocien el estado con la comunicación entre el cliente y el servidor.

- **Worker Thread (Thread trabajador):** Mejora la productividad y minimiza la latencia media.
- **Callback (Retrollamada):** Permite que un cliente se registre en un servidor para ciertas operaciones. De esta forma, el servidor puede notificar al cliente cuando la operación ha finalizado.
- **Succesive Update (Actualización Sucesiva**): Ofrece a los clientes una forma de recibir actualizaciones continuas.
- **Router (Encaminador):** Desacopla múltiples fuentes de información de los objetos de esa información.
- **Transaction (Transacción):** Agrupa una colección de métodos de forma que todos ellos finalicen correctamente o fallen de forma colectiva.

### 1.6.3 GRASP: Patrones de Principios Generales para Asignar Responsabilidades

En diseño orientado a objetos, **GRASP** son patrones generales de software para asignación de responsabilidades, es el acrónimo de "General Responsibility Assignment Software Patterns". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

Los patrones **GRASP** básicos se refieren a cuestiones y aspectos fundamentales del diseño, ellos son:

**Experto:**

Asignar una responsabilidad al experto de información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Expresa simplemente la intuición de que los objetos hacen cosas relacionadas con la información que poseen. Este patrón permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento. Este patrón propicia además que el comportamiento se distribuya entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas, o sea que además brinda soporte a una alta cohesión.

**Creador:**

Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Este patrón brinda soporte a un bajo acoplamiento.

**Bajo Acoplamiento:**

Este es un principio que se debe recordar siempre que se vaya a diseñar algo. Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también reutilizables, que acrecientan la oportunidad de una mayor productividad.

**Alta cohesión:**

Este es también un principio que se debe tener en cuenta en todas las decisiones de diseño. Grady Booch señala que se da una alta cohesión cuando los elementos de un componente, una clase por ejemplo, "colaboran para producir algún comportamiento bien definido". Una clase de alta cohesión posee un número relativamente pequeño de responsabilidades, con una importante funcionalidad relacionada y poco trabajo por hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Este patrón mejora la calidad y facilidad con que se puede entender el diseño, se genera un bajo acoplamiento y permite fomentar la reutilización.

**Controlador:**

La mayor parte de los sistemas reciben eventos de entrada externa[9], los cuales generalmente incluyen una interfaz gráfica para el usuario (IGU) operado por una persona. En estos casos hay que elegir un controlador[10] que maneje esos eventos de entrada. Este patrón se utiliza porque las operaciones del sistema deberían manejarse en la capa de dominio de los objetos, y no en las de interfaz, presentación o aplicación. (Larman, 1999)

## *1.7* Análisis y diseño en RUP

Un flujo de trabajo es una secuencia de actividades que producen un resultado de valor observable. Una enumeración de los roles, actividades y artefactos no constituye un proceso, se necesita una forma de escribir, una forma de describir secuencias significativas de actividades que produzcan un resultado válido y que muestre la interacción entre los roles que

participan.

El análisis y diseño como se mencionó anteriormente es uno de los 9 flujos de trabajo de la metodología RUP y será estudiado en este epígrafe.

El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema. El análisis consiste en obtener una visión del sistema que se preocupa de ver QUÉ hace, de modo que sólo se interesa por los requisitos funcionales. Por otro lado, el diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, en definitiva CÓMO cumple el sistema sus objetivos. El diseño debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. De hecho, cuando la precisión del diseño es muy grande, la implementación puede ser hecha por un generador automático de código.

El objetivo de la Disciplina Análisis y Diseño es: Transformar los requisitos en el Diseño del Futuro Sistema. En la fase de inicio este flujo de trabajo se preocupa por establecer si el sistema como se prevé es factible y por la evaluación de tecnologías potenciales para la solución. Al inicio de la fase de elaboración este flujo de trabajo se centra en crear una arquitectura inicial para el sistema, proporcionando un punto de partida para el principal trabajo de análisis. Si la arquitectura ya existe pues se refina y se analiza el comportamiento y se crea un conjunto inicial de elementos que proporcionarán el comportamiento adecuado. Después de identificar los elementos iniciales, se refinan. El diseño de componentes produce una serie de componentes que proporcionan el comportamiento adecuado para satisfacer los requerimientos. Si el diseño incluye una base de datos el diseño de la misma ocurre en paralelo. El resultado es un conjunto inicial de componentes que se refinarán más aún en la implementación.

**Este flujo tiene como propósito:**

- Transformar los Requisitos, en el Diseño del futuro Sistema.
- Evolucionar una Arquitectura Robusta para el Sistema.
- Adaptar el Diseño para que coincida con el entorno de implementación y los requisitos No Funcionales.

**Roles**

Un rol es la definición del comportamiento y las responsabilidades de un individuo o grupo de individuos que trabajan juntos como un equipo, dentro del contexto de una organización de desarrollo de software.

Los roles no son individuos sino que describen cómo los individuos deben comportarse en el negocio y las responsabilidades de cada uno. Un individuo puede desarrollar varios roles y un rol puede ser desempeñado por varios individuos. (Corporation.)

### 1.7.1 Diseñador

El diseñador es el responsable de diseñar el sistema, dentro de las restricciones de los requerimientos, la arquitectura y el proceso de desarrollo para el proyecto. El diseñador identifica y define las responsabilidades, operaciones, atributos y relaciones de los elementos de diseño. Es quien asegura que el diseño es consistente con la arquitectura del software, y que tiene un nivel de detalle tal que la implementación puede ser realizada.

El diseñador debe tener un conocimiento sólido de:

- Requerimientos del sistema.
- Arquitectura del sistema.
- Técnicas de diseño de software, incluyendo técnicas de análisis y diseño orientadas a objetos, y el Lenguaje Unificado de Modelado (UML).
- Tecnologías con las que el sistema será implementado.
- Directrices del proyecto sobre las formas en que el diseño se refiere a la implementación, incluyendo el nivel de detalle que se espera en el diseño antes de que la implementación pueda proceder. (Corporation.)

### 1.7.2 Analista

Entre los grupos de roles que define RUP se encuentra el de los analistas. Este está integrado por:

- Analista de procesos de negocio.
- Diseñador del negocio.
- Analista de sistema.
- Especificador de requisitos.

El analista es un miembro del equipo de proyecto que es responsable de elicitar e interpretar

las necesidades de los stakeholders, y comunicar esas necesidades al equipo completo. (Corporation.)

### Analista de sistemas

El analista de sistemas realiza y coordina la elicitación de requerimientos y el modelo de casos de uso, exponiendo la funcionalidad del sistema y delimitando el sistema.

Habilidades del analista de sistemas:

- Un experto en la identificación y entendimiento de los problemas y las oportunidades. Esto incluye la capacidad de articular las necesidades que están asociadas con el problema clave que hay que resolver o la oportunidad de realización.
- Debe tener capacidades de comunicación por encima de la media.
- Debe ser un buen facilitador.
- Debe tener amplio conocimiento del negocio y de las tecnologías que se usan para el desarrollo de software.
- Debe tener la habilidad de absorber y entender información nueva rápidamente.
- Debe estar dispuesto a colaborar con todos los miembros del equipo.

## 1.7.3 Artefactos

Las actividades tienen artefactos de entrada y salida. Un artefacto es un producto de trabajo del proceso: los roles usan artefactos para desarrollar actividades, y producen artefactos en el transcurso de la realización de las actividades. (Corporation.)

### Artefactos del análisis y diseño

- Documento de arquitectura de software
- Modelo de despliegue
- Modelo de análisis
- Modelo de diseño
- Prototipo de interfaz de usuario
- Modelo de datos (Corporation.)
- Modelo de análisis

De todos estos artefactos serán desarrollados como parte de la propuesta de solución el

modelo de análisis y el modelo de diseño.

### 1.7.4 Modelo de Análisis

El modelo de análisis es utilizado fundamentalmente por los desarrolladores para comprender cómo debería darse forma al sistema, es decir, como debería ser diseñado e implementado. Este modelo sirve como una primera aproximación al diseño. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

### 1.7.5 Modelo de Diseño

El Modelo de Diseño es un modelo de objetos que describe la realización física de los casos de uso, sirve como una abstracción del modelo de implementación y su código fuente. El modelo de diseño es utilizado como una entrada esencial a las actividades de implementación y pruebas. (Corporation.)

## 1.8 Estudio de las principales herramientas para la generación de reportes en el mundo.

En nuestros días el uso de las tecnologías se ha hecho imprescindible para las grandes empresas del mundo, ha sido de gran necesidad la utilización de herramientas que le permitan a los desarrolladores evaluar el resultado de su trabajo, obtener una entrega periódica de información en formatos preestablecidos, de fácil entendimiento para los usuarios y que involucre los aspectos relevantes para poder realizar el proceso de toma de decisiones en forma rápida, segura y eficiente, o sea, una herramienta que le permita a las organizaciones acceder, dar formato y distribuir fácilmente la información a empleados, clientes y asociados, estas herramientas son las llamadas generadores de reportes que permiten imprimir informes diseñados por los usuarios en segundos.

En el mundo actual existen muchas de estas herramientas, por solo mencionar algunas de las más usadas se realizó el estudio del estado del arte de algunas de éstas mencionando sus aspectos más relevantes:

### 1.8.1 Crystal Reports

Tipo: Paquete y/o Sistema

Plataforma: Propietaria

Características fundamentales:

- Soporta gráficos, imágenes y sub-reportes
- Soporta una amplia variedad de orígenes de datos
- Cuenta con un lenguaje de programación propio y soporta sentencias de Visual Basic
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc)
- Da soporte para Web y aplicaciones de escritorio

Otras

- Viene integrado a algunas herramientas de desarrollo como Visual studio.NET.
- Cuenta con años de experiencia y desarrollo.

Desventajas:

- Es propietario y las licencias son costosas.
- Para modificar el reporte es preciso recompilar el proyecto donde se encuentra.

### 1.8.2 Active Reports

Tipo: Sistema

Plataforma: Propietaria

Características fundamentales:

- Soporta gráficos, imágenes y sub-reportes
- Soporta una amplia variedad de orígenes de datos
- Cuenta con un lenguaje de programación propio y soporta sentencias de Visual Basic.
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc)
- Da soporte para Web y aplicaciones de escritorio
- Permite la adición de nuevos reportes sin necesidad de recompilar la aplicación

Otras:

- Cuenta con años de experiencia y desarrollo.

Desventajas:

Es propietario y las licencias son costosas.

### 1.8.3 Reporting Service

Tipo: Paquete

Plataforma: Propietaria

Características fundamentales:

- Es una potente herramienta para generar reportes a partir de servidores de Microsoft SQL Server
- Utiliza el formato estándar XML
- Integrada en todas las versiones de SQL 2005
- Se muestra en el visor integrado distribuido con el Framework 2.0
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc)
- Da soporte para Web y aplicaciones de escritorio

Desventajas:

- Es propietario
- Dependiente de la plataforma .NET
- Fuente de datos restringida a servidores de Microsoft SQL Server

### 1.8.4 Jasper Reports

Tipo: Paquete y/o Sistema

Plataforma: Libre

Características fundamentales:

- Multiplataforma
- Soporta imágenes y sub-reportes
- Soporta varios orígenes de datos
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc.)
- Da soporte para Web y aplicaciones de escritorio
- Se integra con el JfreeChart (librería libre para la generación de graficas escrita en Java)

Desventajas:

- Dependiente de la plataforma Java
- Complejo de usar
- Deficiente robustez y eficacia para reportes complejos.

### 1.8.5 DynamicJasper

Tipo: Paquete

Plataforma: Libre

Características fundamentales:

- Reduce la complejidad del Jasper Report para reportes simples y de mediana complejidad.
- Mantiene las principales características del Jasper Report ya que trabaja directamente con él.
- Añade cierto grado de dinamismo a los reportes en tiempo de ejecución.

Desventajas:

- Dependiente de la plataforma Java.
- Deficiente robustez y eficacia para reportes complejos.

### 1.8.6 Actuate Enterprice Reporting

Tipo: Sistema

Plataforma: Propietario

Características fundamentales:

- Flexible en cuanto al formato y diseño
- De fácil manejo y mantenimiento
- Potente para reportes internos

Desventajas

- Es propietario
- Diseñado solo para la Web

### 1.8.7 Agata Report

Tipo: Paquete y/o sistema

Plataforma: Libre

Características fundamentales:

- Multiplataforma (Windows y Linux)

- Soporta gráficos, imágenes y sub-reportes
- Soporta varios orígenes de datos
- Exporta salidas en diferentes formatos (PDF, XML, HTML PostScript, plain text, HTML, XML, PDF o CSV, StarCalc, Excel)
- Es fácil de instalar y ejecutar
- Es extremadamente flexible.

Desventajas:

- Dependiente de la plataforma PHP sobre la que corre directamente y por tanto:
- Diseñado solo para la Web usando PHP

### 1.8.7 Report Manager

Tipo: Paquete y/o Sistema

Plataforma: Libre

Características fundamentales:

- Multiplataforma (Windows y Linux).
- Soporta gráficos, imágenes y sub-reportes.
- Soporta varios orígenes de datos.
- Soporte para líneas de comandos.
- Integra un servidor de reportes.

Desventajas:

- Viene especialmente diseñado para usar con Delphi, C++ Builder y Kylix.
- Su interfaz gráfica es poco intuitiva y poco amigable.
- Se hace difícil dominarlo y comprender su funcionamiento.

### 1.8.8 MoReport

Basado en las librerías GTK y liberado con licencia GNU/GPL.

Características fundamentales:

- soporte para base de datos PostgreSQL.
- uso del estándar XML como formato de entrada, y salidas en PostScript. Actualmente se encuentra en su versión 0.1.

### 1.8.9 Quick Report

Tipo: Paquete

Plataforma: Libre

Características fundamentales:

- Soporta gráficos, imágenes y sub-reportes
- Soporta varios orígenes de datos
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc)
- Soporta herencia visual

Desventajas:

- Diseñado para usar con Delphi y C++ Builder
- Su interfaz gráfica es poco intuitiva y poco amigable
- Se hace difícil dominarlo y comprender su funcionamiento

### 1.8.10 FastReport

Tipo: Paquete y/o Sistema

Plataforma: Libre

Características fundamentales:

- Multiplataforma (Windows y Linux)
- Soporta gráficos, imágenes y sub-reportes
- Soporta varios orígenes de datos
- Exporta salidas en diferentes formatos (PDF, XML, HTML, etc)
- Soporta herencia visual

Desventajas:

- Diseñado para usar con Delphi y C++ Builder
- Su interfaz gráfica es poco intuitiva y poco amigable
- Se hace difícil dominarlo y comprender su funcionamiento

### 1.8.11 Smart Report Maker

Generador de Reportes PHP/MySQL de muy fácil uso, que le permite crear y administrar un número ilimitado de Reportes MySQL basados en tablas o consultas.

Características fundamentales:

- Cuenta con una interface de asistente amigable que le permite seleccionar únicamente los campos que usted desea se presenten en su reporte.
- Configura múltiples niveles de agrupamiento.
- Ordena los datos de forma ascendente o descendente y tiene la capacidad de exportar los reportes a archivos de formatos PDF, XML o CSV.
- La apariencia del reporte generado es totalmente personalizable, se puede seleccionar el diseño y la hoja de estilos en cascada que guste.

En nuestra universidad el proyecto del Menpet creó el generador de reportes:

### 1.8.12 Olympia

Es una aplicación web que tiene como objetivo generar reportes de forma rápida, interactiva y con una amplia gama de alternativas para los usuarios. La extensión en su uso puede estandarizar la generación de reportes en diferentes aplicaciones independientemente del sistema gestor de base de datos que utilicen ya sea MySQL o PostgreSQL.

El sistema permite a los usuarios, entre otras opciones, abstraerse de los conocimientos relacionados con los gestores de bases de datos siempre que las consultas generadas se implementen en SQL estándar, agilizar la toma de decisiones, generar reportes en varios formatos y con gran variedad de opciones en su diseño, marcando una diferencia entre los reportes tradicionales y los reportes dinámicos, objetos de este producto.

**Plataforma:** Libre

**Características:**

- Desarrollado con php y el framework symfony.
- Es un sistema que se puede aplicar en cualquier aplicación que necesite un generador de reportes dinámico.
- Permite la administración de reportes.
- Posee un diseñador de plantillas web.
- Desarrollado con tecnologías open source.

También se encuentran algunos proyectos productivos que utilizan algunos reporteadores dinámicos, tal es el caso de ERP que utiliza phpReport aunque se piensa que posteriormente utilice El Olympia y se encuentra Akademos que utiliza el CrystalReport.

## *1.9 Conclusiones parciales*

Para realizar una optima construcción de un sistema para la generación de reportes, este capítulo se ha dedicado a realizar un estudio a las diferentes temáticas que ayuden a guiar el proceso de desarrollo del software tales como, metodologías de desarrollo de software, herramientas Case y lenguajes de modelado. De este estudio resultó la selección de la metodología RUP, el Lenguaje Unificado de Modelado y la herramienta CASE Visual Paradigm for UML. Se realizó el estudio del estado del arte de los patrones de casos de uso y de diseño, para posteriormente seleccionar los que más se ajusten a la solución. Se estudió además las características de las principales herramientas generadoras de reporte para de esta forma lograr que la herramienta que se quiere desarrollar logre solucionar el problema que inició esta investigación.

# CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA. ANÁLISIS Y DISEÑO.

## *2.1 Introducción*

Este capítulo constituye la propuesta de solución para estandarizar el proceso de generación de reportes en la Facultad 3. Para describir la solución se realiza el modelo de dominio, un glosario de términos para definir los diversos conceptos que serán utilizados, así como las relaciones que entre estos conceptos se establecen, en aras de logar la utilización de un vocabulario común. Se realiza la especificación de los requerimientos funcionales y no funcionales que debe presentar la solución a construir. Se determinan los casos de uso del sistema y los actores que interactuarán con éste, elaborándose una descripción en formato expandido de cada uno de los casos de uso, así como el análisis y diseño del sistema con los diagramas correspondientes a este flujo, conjuntamente con esto se exponen los patrones de diseño empleados en la solución.

## *2.2 Modelo Conceptual o Modelo de Dominio*

El análisis orientado a objetos tiene por finalidad estipular una especificación del dominio del problema y los requerimientos desde la perspectiva de la clasificación por objetos y desde el punto de vista de entender los términos empleados en el dominio. Para descomponer el dominio del problema hay que identificar los conceptos, los atributos y las asociaciones del dominio que se juzgan importantes. El resultado puede expresarse en un modelo conceptual, el cual se muestra gráficamente en un grupo de diagramas que describen los conceptos (objetos).

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema. Representa conceptos del mundo real, no de los componentes de software. Este constituye el artefacto más importante a crear durante el análisis orientado a objetos

El modelo de dominio se representa en UML con un Diagrama de Clases en los que se muestra:

- conceptos u objetos del dominio del problema: clases conceptuales
- asociaciones entre las clases conceptuales

- atributos de la clase conceptuales

En el Modelo de Dominio no se muestra comportamiento. Las clases conceptuales pueden tener atributos pero no métodos.

Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión).

Las clases del dominio o clases conceptuales aparecen en tres formas típicas:

- Objetos del negocio: Representan cosas que se manipulan en el negocio.
- Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento.
- Sucesos que ocurrirán o han ocurrido. (Rumbaugh, y otros, 2000)

### 2.2.1 ¿Por qué realizar un modelo de dominio?

Teniendo en cuenta que la Universidad es un centro de estudios y producción y que en ella existen varios proyectos productivos en los que se desarrollan sistemas que en su mayoría tienen como un requisito indispensable la gestión de la información, se hace necesario realizar el análisis y diseño para permitir una posterior implementación de una herramienta que posibilite la obtención de la información, en forma de reportes. Actualmente, en la Universidad se utilizan herramientas conocidas como el Crystal Report y el Active Report, o se crean sistemas específicos para un proyecto determinado, e incluso existen proyectos que no cuentan con ningún tipo de herramienta para realizar este proceso, esto implica un coste de tiempo y esfuerzo innecesario para los desarrolladores.

Con esta investigación se pretende proporcionar los artefactos necesarios para una posterior implementación del sistema que pueda ser utilizado por los diferentes proyectos productivos. Este sistema presenta dos aspectos novedosos fundamentales:

1. Independiente del lenguaje de programación utilizado.
2. Diseñador gráfico del reporte en dos modos: Modo Avanzado para usuarios expertos en informática; y Modo Básico para usuarios no experimentados en el uso y manejo de herramientas informáticas como clientes y usuarios finales.

Esta herramienta presentará además, las siguientes características:

- Basada en Software Libre.
- Multiplataforma
- Conexión a múltiples bases de datos.
- Número ilimitado de sub-reportes y secciones y grupos.

- Editor de expresiones.
- Exportar a múltiples formatos (PDF, HTML, RTF).
- Herencia entre reportes.
- Gráficos, imágenes, textos y líneas
- Basado en la filosofía WYSIWYG.

Entre otras. Para un listado completo de las características del sistema ver **Anexo 3**

Muchas de las herramientas que en estos momentos se usan en la Universidad son propietarias, lo que implica el pago de costosas licencias por su utilización. Por ello, y como parte del proceso de migración que se quiere llevar a cabo en nuestro país y particularmente en la Universidad, una de las características fundamentales de la herramienta que se quiere construir es que sea libre. Se pretende que el sistema facilite y estandarice el proceso de generación de reportes en la Facultad 3, permitiendo obtener información valiosa, de manera práctica, rápida y sencilla.

Se ha decidido realizar un modelo de dominio ya que es una alternativa apropiada dado el escenario del problema, pues el objetivo primario de este sistema es la gestión y presentación de información. Además de que no se considera necesario un modelamiento completo del negocio, las fronteras del negocio no se pueden determinar con claridad y los procesos del negocio no están claramente definidos.

## *2.3 Modelo de Dominio Propuesto*

El objetivo del modelado del dominio es contribuir a la comprensión del contexto del sistema, y por lo tanto también contribuir a la comprensión de los requisitos del sistema que se desprenden de este contexto. El modelado de dominio debe contribuir a una comprensión del problema que se supone que el sistema resuelve en relación a su contexto. A continuación la figura muestra el modelo de dominio propuesto:

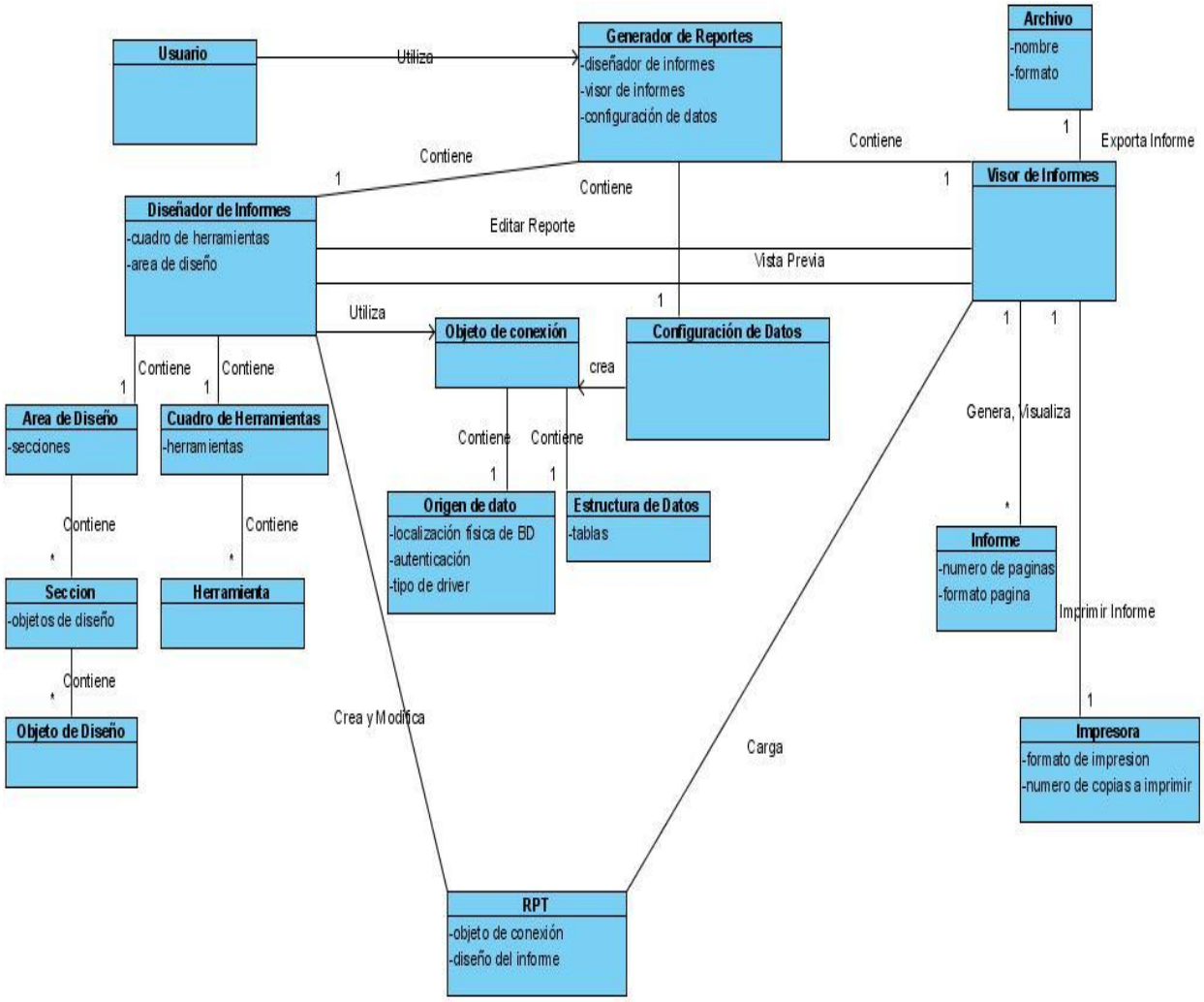



**Figura 2.1 Modelo del Dominio Propuesto**

## *2.4 Glosario de Términos*

El glosario de términos junto al Modelo de Dominio ayuda a los usuarios, clientes, desarrolladores y otros interesados a utilizar un vocabulario común. La terminología común es necesaria para compartir el conocimiento con los otros. El glosario es muy útil para alcanzar un entendimiento entre los desarrolladores, relativo a la definición de los diversos conceptos y nociones, y para reducir en general, el riesgo de confusiones. Para construir un sistema software de cualquier tamaño, los ingenieros de hoy en día deben “fundir “el lenguaje de todos los participantes en uno solo consistente.

En el Glosario de Términos se definen términos comunes importantes que utilizan los analistas al describir el sistema. Es un artefacto muy intuitivo para usarlo con terceras personas externas, como usuarios y clientes. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

Para consultar el Glosario de Términos **Ver Anexo 2**

## *2.5 Modelo del sistema*

En este epígrafe se especifican los requisitos tanto funcionales como no funcionales que tendrá el subsistema que se quiere construir, así como el modelamiento del subsistema en términos de casos de uso.

### 2.5.1 Requerimientos del sistema

La IEEE Standard Glossary of Software Engineering Terminology define un requerimiento como:

- Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
- Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

El propósito fundamental de la captura de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema. (Jacobson, Ivar, Rumbaugh, Booch, & Booch, 2004). Para capturar los requisitos se utilizaron las siguientes técnicas: tormenta de ideas, casos de uso, comparación de terminología, estudio de otros sistemas y entrevistas **(Ver Anexo 4).** Esta última técnica fue aplicada a los desarrolladores de reportes de algunos proyectos productivos en la UCI, cuya experiencia en la generación de reportes oscila entre 1 y 4 años. Los resultados arrojados fueron los siguientes:

**Características importantes en un sistema generador de reportes:** Soporte para gráficos, imágenes, sub-reportes, orígenes de datos diversos, líneas y textos.

**Características sugeridas para la herramienta propuesta:** Editor de fórmulas, herencia entre reportes, Vista Previa

### 2.5.1.1 Requerimientos funcionales

Son capacidades o condiciones que el sistema debe cumplir.

Para cumplir los objetivos de este sistema el mismo debe ser capaz de:

R 1: Seleccionar Gestor de Base de Datos.
R 2: Revisar Historial.
R 3: Autenticarse.
R 4: Ver Estructura de la Base de Datos.
R 5: Seleccionar Estructura de la Base de Datos.
R 6: Ver Conexión Actual.
R 7: Seleccionar Base de Datos.
R 8: Crear Origen de Datos.
R 9: Gestionar Origen de Datos.
R10: Cambiar de Conexión.

### 2.5.1.2. Requerimientos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

- **Requerimientos de software**
  - Sistema Operativo Microsoft Windows 2000 o superior; Sistema Operativo Linux con ambiente grafico KDE o GNOME.
  - Plataforma Mono 1.9 o superior con GTK# 2.10 o superior.
- **Requerimientos de hardware**
  - Se requiere un mínimo de 256 MB de RAM, 512 MB recomendado, y 1.3 GHz de velocidad de procesamiento.
  - Requerimientos de apariencia e interfaz externa
  - La interfaz debe ser sencilla y amigable de manera que potencie la comodidad del usuario para su trabajo.
  - Las opciones más usadas presentarán vías rápidas y cómodas de invocarse.

- Debe presentar una interfaz especial para usuarios no expertos en informática como usuarios finales y clientes, que sea muy amigable y fácil de trabajar y, que satisfaga las necesidades básicas de diseño de reportes.
- Los informes generados seguirán la norma WYSIWYG (What You See Is What You Get)

**Requerimientos de usabilidad**

- La interfaz para usuarios no expertos la podrá usar cualquier usuario con un mínimo de capacitación y experiencia en el trabajo con herramientas informáticas.
- La interfaz para usuarios expertos podrá ser usada por cualquier desarrollador, personal de soporte u otra persona con vasta experiencia en el uso de herramientas informáticas.
- Contendrá un manual de usuario y una ayuda que instruirán al usuario en el trabajo con el sistema.

**Requerimiento de portabilidad**

- Una de las mayores ventajas que tendrá el sistema es su portabilidad ya que el mismo podrá correr en cualquier plataforma, Windows o Linux.

**Restricciones de diseño e implementación**

- El sistema será implementado en la plataforma .NET para luego ser migrado a la plataforma MONO.

**Rendimiento**

- El sistema debe requerir un consumo mínimo de recursos.
- Debe tener tiempos de respuesta rápidos garantizando de esta forma la agilidad del sistema.

### 2.5.2. Modelo de casos de uso del sistema.

Un modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones.

El modelo de casos de uso permite que los desarrolladores y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. El modelo de casos de uso describe lo que hace el sistema para cada tipo de usuario. Cada usuario se representa mediante uno o más actores. (Jacobson, Ivar, Rumbaugh & Booch, 2004)

### 2.5.2.1 Actores del sistema.

Los actores representan terceros fuera del sistema que colaboran con el sistema. Al identificar los actores del sistema se identifica el entorno externo del sistema.


**Figura 2.2.** Actor del sistema

| Actor del Sistema | Justificación |
|---|---|
| Usuario | Generaliza a todas las personas dentro de un proyecto productivo que interactúan con el generador de reportes. |

### 2.5.2.2. Casos de uso del sistema

**Figura 2.3** Estereotipo de Caso de uso del sistema

Cada forma en que los actores usan el sistema se representa con un caso de uso. Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. Un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia. (Jacobson, Ivar, Rumbaugh & Booch, 2004). Los casos de uso son el componente clave del modelado. Su propósito es ilustrar como un sistema permite a un actor cumplir una meta, ilustrando todos los posibles caminos apropiados que ellos pueden tomar para cumplirla, así como las situaciones que podrían hacerlo fallar.

**Casos de uso determinados para satisfacer los requerimientos funcionales del sistema:**

Resumen del Caso de Uso 1

| CU-1 | Gestionar Estructura de la Base de Datos |
|---|---|
| Actor | Usuario |
| Descripción | El caso de uso se inicia cuando el usuario desea un Nuevo origen de Datos o gestionar uno ya existente. |
| Referencia | R1, R4, R5, R7, R8, R9, R10 |

Resumen del Caso de Uso 2

| CU-2 | Autenticarse |
|---|---|
| Actor | Usuario |
| Descripción | El caso de uso inicia cuando el usuario desea acceder a la Base de Datos. |
| Referencia | R3, R1, R8 |

Resumen del Caso de Uso 3

| CU-3 | Mostrar Historial |
|---|---|
| Actor | Usuario |
| Descripción | El caso de uso inicia cuando el usuario desea ver el historial. |
| Referencia | R2. |

Resumen del Caso de Uso 4

| CU-4 | Ver Conexión Actual |
|---|---|
| Actor | Usuario |
| Descripción | El caso de uso inicia cuando el usuario desea revisar los usuarios que están conectados en ese momento a la Base de Datos. |
| Referencia | R6 |

### 2.5.2.3. Diagrama de Casos de uso del sistema.

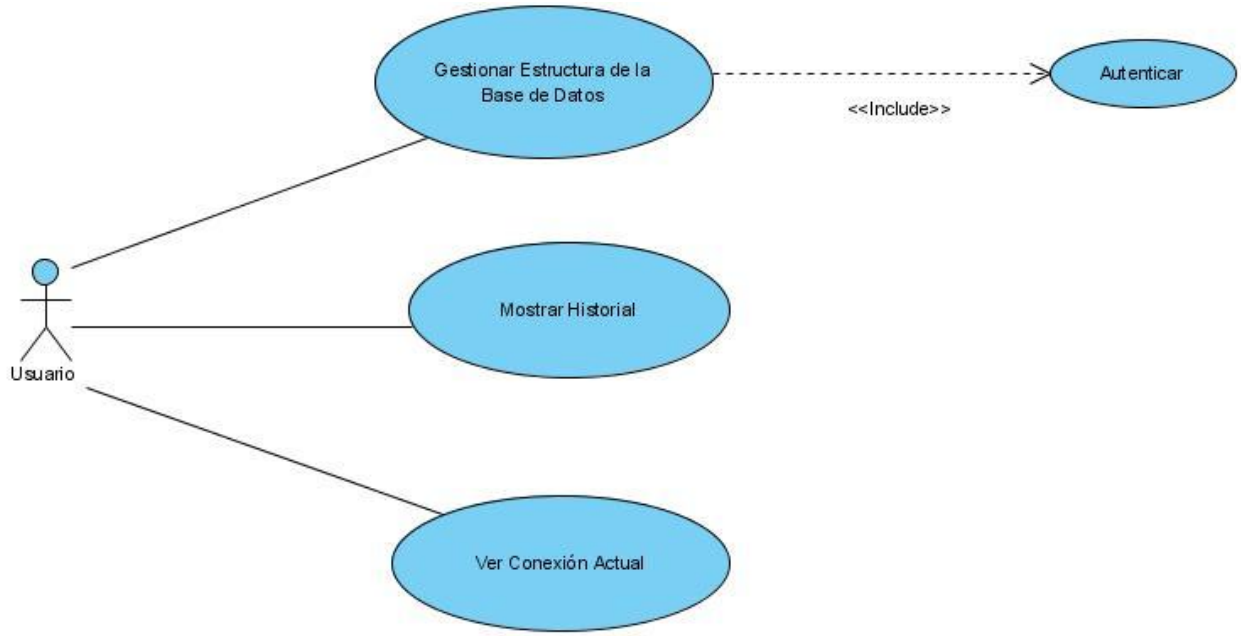



**Figura 2.4 Diagrama de Casos de Uso del Sistema**

- Para estructurar el diagrama de casos de uso del sistema se tuvieron en cuenta los siguientes patrones de casos de uso: CRUD completo, Extensión concreta o inclusión: Inclusión.

### 2.5.2.4 Especificación textual de los casos de uso.

Un caso de uso consiste principalmente de una especificación textual (llamada Especificación del caso de uso), que contiene una descripción del flujo de eventos, describiendo la interacción entre actores y el sistema. La especificación contiene además otras informaciones, como son precondiciones, poscondiciones, requerimientos especiales y escenarios claves.

La especificación de los casos de uso contiene las propiedades textuales de los casos de uso. Es utilizada junto a una herramienta de gestión de requisitos para especificar y marcar los requerimientos dentro de las propiedades de los casos de uso. (Corporation.) Para organizar las descripciones de los casos de uso se utilizó el patrón Cohesión: Rehúso interno, Escenario más Fragmentos, Alternativas Exhaustivas Íntegras.

**Descripción detallada del CU Gestionar Estructura de la Base de Datos**

| Caso de Uso: | Gestionar Estructura de la Base de Datos |
|---|---|
| Actores: | Usuario |
| Resumen: | El caso de uso se inicia cuando el usuario desea un Nuevo origen de Datos o Gestionar uno ya existente. |
| Precondiciones: | Que se encuentre en alguna base de datos. |
| Referencias: | R1, R4, R5, R7, R8, R9 |
| Prioridad: | Crítico |

| Flujo Normal de Eventos | |
|---|---|
| Acción de Actor | Respuesta del Sistema |
| 1- El usuario accede a la Interfaz Configuración de Datos. | 2- El sistema le va a mostrar todas las opciones que tiene esta interfaz. |
| 3- El usuario selecciona la opción Nuevo Origen de Datos o Gestionar Origen de datos, Si es:<br>a) Nuevo Origen de Datos: Ver Sección Nuevo Origen de Datos.<br>b) Gestionar Origen de datos: Ver Sección Gestionar Origen de datos | |
| Prototipo Interfaz | |
| Ver Anexos Figura 1 | |
| Sección Nuevo Origen de Datos | |
| 1- El usuario selecciona la opción Nuevo Origen de Datos. | 2- El sistema le va a mostrar una interfaz con los gestores de base de datos que tiene para que seleccione el que desea |
| 3- El usuario va a seleccionar el que desea | 4- El sistema le pide que se autentique Ver CU Autenticarse.<br>Si la BD existe va al catalogo de la base de |

| | |
|---|---|
| | datos y le pide tablas de la BD, Atributos, Vistas, Funciones y Procedimientos, crea el objeto de conexión y le muestra al usuario una interfaz con todas las estructuras de las tablas para seleccionar |
| 5- El usuario selecciona las tablas que desea y pulsa el botón Salvar. | 6- El sistema guarda el objeto en la BD. |
| Flujo Alterno 1 | |
| | 4.1- El sistema le muestra un error al usuario que no puede efectuar la conexión |
| Flujo Alterno 2 | |
| 5.1- El usuario selecciona las tablas que desea y pulsa el botón Diseñar. | 5.2- El sistema le envía al diseñador el objeto que se creo. |
| Prototipo Interfaz Sección Nuevo Origen de Datos | |
| Ver Anexos Figura 2 | |
| Sección Gestionar Origen de Datos | |
| 1- El usuario selecciona la opción Gestionar Origen de Datos. | 2- El sistema le muestra una interfaz con la lista de nombres de los orígenes de datos existentes. |
| 3- El usuario selecciona con la que desea trabajar y pulsa el botón aceptar. | 4- El sistema le va a mostrar las tablas del origen de datos que seleccionó. |
| Prototipo Interfaz Sección Gestionar Origen de Datos | |
| Ver Anexos Figura 3 | |

**Descripción detallada del CU Autenticarse.**

| Caso de Uso: | Autenticarse |
|---|---|
| Actores: | Usuario |
| Resumen: | El caso de uso inicia cuando el usuario desea acceder a la Base de Datos. |
| Precondiciones: | Que se encuentre la Base de Datos. |
| Referencias: | R3, R1, R8 |
| Prioridad: | Crítico |
| Flujo Normal de Eventos | |
| Acción de Actor | Respuesta del Sistema |

| | |
|---|---|
| 1- El usuario accede a la interfaz principal Configuración de Datos. | 2- El sistema le muestra al usuario las opciones que tiene la interfaz. |
| 3- El usuario selecciona el tipo de Base de Datos a la cual quiere conectarse. | 4- El sistema le muestra una interfaz con las bases de datos de ese tipo existentes. |
| 5- El usuario selecciona la base de datos que desea utilizar. | 6- El sistema le muestra una ventana de autenticación según el tipo de Base de Datos seleccionada.<br><br>Ver Secciones del CU Autenticarse. |
| **Sección Autenticarse para Postgres** | |
| | 1- El sistema le muestra una ventana con los campos a llenar para la autenticación:<br><br>- Nombre del servicio<br>- Nombre de la cuenta<br>- Dominio<br>- Contraseña<br>- Verificación de Contraseña<br><br>Así como un botón desplegable con las bases de datos de este tipo que existen. |
| 2- El usuario introduce los datos que le piden y selecciona la base de datos que desea. | 3- El sistema verifica la validez de los datos y la existencia de la base de datos seleccionada. |
| **Flujo Alterno** | |
| | 3.1- El sistema le muestra un mensaje de error pidiéndole que verifique los datos introducidos |
| **Prototipo Interfaz Sección Autenticarse para Postgres** | |
| Ver Anexos Figura 4 | |
| **Sección Autenticarse para Oracle** | |
| | El sistema le muestra una ventana con los campos a llenar para la autenticación:<br>-Usuario<br>-Contraseña<br>-Verificación de Contraseña |

| | |
|---|---|
| | -Nombre de la Base de Datos.<br><br>Así como un botón desplegable con las bases de datos de este tipo que existen. |
| 1- El usuario introduce los datos que le piden y selecciona la base de datos que desea. | 2- El sistema verifica la validez de los datos y la existencia de la base de datos seleccionada. |
| Flujo Alterno | |
| | 3.1- El sistema le muestra un mensaje de error pidiéndole que verifique los datos introducidos |
| Prototipo Interfaz Sección Autenticarse para Oracle | |
| Ver Anexos Figura 5 | |
| Sección Autenticarse en MySQL | |
| | 1- El sistema le muestra una ventana con los campos a llenar para la autenticación:<br><br>-Servidor.<br>-Usuario<br>-Contraseña<br>-Rectificación de Contraseña<br>-Puerto<br>-Tipo de Servidor<br><br>Así como un botón desplegable con las bases de datos de este tipo que existen. |
| 2- El usuario introduce los datos que le piden y selecciona la base de datos que desea. | 3- El sistema verifica la validez de los datos y la existencia de la base de datos seleccionada. |
| Flujo Alterno | |
| | 3.1- El sistema le muestra un mensaje de error pidiéndole que verifique los datos introducidos. |
| Prototipo Interfaz Sección Autenticarse para MySQL | |
| Ver Anexos Figura 6 | |
| Sección Autenticarse en Access | |
| | 1- El sistema le muestra una ventana con los campos a llenar para la autenticación:<br><br>-Nombre de la Base de Datos.<br>-Contraseña de la Base de Datos.<br>-Usuario.<br>-Contraseña.<br>-Rectificación de la contraseña.<br>Así como un botón desplegable con las |

| | |
|---|---|
| | bases de datos de este tipo que existen. |
| 2- El usuario introduce los datos que le piden y selecciona la base de datos que desea. | 3- El sistema verifica la validez de los datos y la existencia de la base de datos seleccionada. |
| Flujo Alterno | |
| | 3.1- El sistema le muestra un mensaje de error pidiéndole que verifique los datos introducidos. |
| Prototipo Interfaz Sección Autenticarse para Access | |
| Ver Anexos Figura 7 | |
| Sección Autenticarse en SQL Server. | |
| | 1- El sistema le muestra una ventana con los campos a llenar para la autenticación:<br><br>-Nombre.<br>-Contraseña.<br>-Rectificación de Contraseña.<br>-Dominio.<br><br>Así como un botón desplegable con las bases de datos de este tipo que existen. |
| 2- El usuario introduce los datos que le piden y selecciona la base de datos que desea. | 3- El sistema verifica la validez de los datos y la existencia de la base de datos seleccionada. |
| Flujo Alterno | |
| | 3.1- El sistema le muestra un mensaje de error pidiéndole que verifique los datos introducidos. |
| Prototipo Interfaz Sección Autenticarse para SQL Server | |
| Ver Anexos Figura 8 | |

**Descripción detallada del CU Mostrar Historial**

| Caso de Uso: | Mostrar Historial |
|---|---|
| Actores: | Usuario |
| Resumen: | El caso de uso inicia cuando el usuario desea ver el historial. |
| Precondiciones: | |
| Referencias: | R2. |

| Prioridad: | Secundario |
|---|---|
| Flujo Normal de Eventos | |
| Acción de Actor | Respuesta del Sistema |
| 1- El usuario accede a la Interfaz Configuración de Datos. | 2- El sistema le va a mostrar todas las opciones que tiene esta interfaz. |
| 3- El usuario escoge la opción mostrar historial y accede a la interfaz. | 4- El sistema le va a pedir que seleccione un rango de fecha para mostrar el historial. |
| 5- El usuario va a introducir los datos que se le pide. | 6- El sistema le va a mostrar todo el historial según la fecha introducida. |
| Flujo Alterno | |
| | 4.1- El sistema le muestra un mensaje de error en el rango de fechas introducido |
| 4.2- Vuelve al paso 3 | |
| Prototipo Interfaz | |
| Ver Anexos Figura 9 | |

**Descripción detallada del CU Ver Conexión Actual**

| Caso de Uso: | Ver Conexión Actual |
|---|---|
| Actores: | Usuario |
| Resumen: | El caso de uso inicia cuando el usuario desea revisar los usuarios que están conectados en ese momento a la Base de Datos. |
| Precondiciones: | |
| Referencias: | R6 |
| Prioridad: | Secundario |
| Flujo Normal de Eventos | |
| Acción de Actor | Respuesta del Sistema |
| 1- El usuario accede a la Interfaz Configuración de Datos. | 2- El sistema le va a mostrar todas las opciones que tiene esta interfaz. |
| 3- El usuario escoge la opción y | 4- El sistema la va mostrar las conexiones |

| | |
|---|---|
| accede a la interfaz Ver Conexión Actual. | que existen en ese momento. |
| Flujo Alterno | |
| | 4.1- El sistema le va a mostrar un aviso de que no hay conexiones en ese momento. |
| Prototipo Interfaz | |
| Ver Anexos Figura 10 | |

## *2.6 Análisis del sistema.*

Durante el análisis se analizan los requisitos que se obtuvieron en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero, incluyendo su arquitectura.

En el análisis podemos razonar más sobre los aspectos internos del sistema. Podemos estructurar los requisitos de manera que nos facilite su comprensión, su preparación, su modificación, y en general su mantenimiento (Rumbaugh, et al., 2004).

### 2.6.1 Modelo de análisis

El lenguaje usado en el análisis se basa en un modelo de objetos conceptual llamado modelo de análisis. El modelo de análisis ayuda a estructurar los requisitos y proporciona una estructura basada en aspectos tales como la flexibilidad ante los cambios y la reutilización. Esta estructura no solo es útil para el mantenimiento de los requisitos como tal, sino que también se utiliza como entrada en las actividades de diseño e implementación. Mediante la conservación de la estructura del modelo de análisis durante el diseño, se obtiene un sistema que debería ser también mantenible como un todo: será flexible a los cambios en los requisitos, e incluirá elementos que pueden ser reutilizados cuando se construyan sistemas parecidos.

#### 2.6.1.1 Realización de caso de uso- análisis

Una realización de caso de uso-análisis es una colaboración dentro del modelo de análisis que describe cómo se lleva a cabo y se ejecuta un caso de uso determinado en términos de clases del análisis y de sus objetos del análisis en interacción. Por tanto, proporciona una traza directa

hacia un caso de uso concreto del modelo de casos de uso.

Una realización de caso de uso-análisis posee una descripción textual del flujo de sucesos, diagramas de clases que muestran sus clases del análisis participante, y diagramas de interacción que muestran la realización de un flujo o escenario particular del caso de uso, en términos de clases del análisis y de sus objetos.

**Diagrama de clases del análisis**

El diagrama de clases del análisis representa las clases del análisis que participan en las realizaciones de los casos de uso-análisis y las relaciones que se establecen entre ellas.

**Clases del análisis**

Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema. Esta abstracción posee las siguientes características:

- Se centra en el tratamiento de los requisitos funcionales.
- Las clases de análisis siempre encajan en uno de tres estereotipos básicos: de interfaz, de control o de entidad.

**Clases de interfaz**

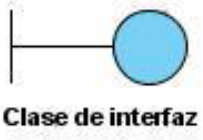



**Figura 2.5.** Estereotipo de la clase de interfaz.

Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores. Esta información a menudo implica recibir y presentar información y peticiones de y hacia los usuarios y los sistemas externos.

**Clases de control**


**Figura 2.6.** Estereotipo de la clase de control

Las clases de control representan coordinación, secuencia, transacciones, y control de otros objetos, y se usan con frecuencia para encapsular el control de un caso de uso en concreto.

**Clases de entidad**


**Figura 2.7.** Estereotipo de las clases de entidad.

Las clases de entidad se utilizan para modelar información que posee una larga vida y que es a menudo persistente. Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto o un suceso del mundo real.

Diagramas de clases del análisis por casos de uso.
**Ver Anexos Figuras 24-29**

### 2.6.1.2 Diagramas de interacción

Los diagramas de interacción representan una vista dinámica del sistema y se pueden clasificar en dos tipos: de colaboración y de secuencia.

En el caso de los diagramas de interacción según (Rumbaugh & Grady, 2000) es preferible en el análisis realizar diagramas de colaboración, ya que el objetivo fundamental es identificar requisitos y responsabilidades sobre los objetos, y no identificar secuencias de interacción detalladas y ordenadas cronológicamente, pues en ese caso sí sería conveniente utilizar los de secuencia.

Diagramas de colaboración del análisis **Ver Anexos Figuras 11-19**

## *2.7 Diseño*

En el diseño se modela el sistema y se encuentra su forma para que soporte todos los requisitos-incluyendo los requisitos no funcionales y otras restricciones –que le suponen. Una entrada esencial en el diseño es el resultado del análisis, esto es el modelo de análisis.

Los propósitos del diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, etc.

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.

- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.

- Ser capaces de visualizar y reflexionar sobre el diseño utilizando una notación común. (Rumbaugh & Grady, 2000).

### 2.7.1 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. El modelo de diseño sirve de abstracción a la implementación, y es de ese modo, utilizado como una entrada fundamental a las actividades de implementación.

### 2.7.1.1 Realización de caso de uso-diseño

Una realización de caso de uso-diseño es una colaboración en el modelo de diseño que describe cómo se realiza un caso de uso específico, y como se ejecuta, en términos de clases de diseño y sus objetos. Proporciona una traza directa a una realización de casos de uso-análisis en el modelo de análisis.

Una realización de caso de uso-diseño tiene una descripción textual del flujo de eventos, diagramas de clases que muestran sus clases de diseño participantes, y diagramas de interacción que muestran las realizaciones de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño.

**Diagrama de clases del diseño**

En este diagrama se representan las clases participantes en las realizaciones de caso de uso, subsistemas y sus relaciones. También puede darse el caso de algunas operaciones, atributos y asociaciones sobre una clase específica que son relevantes.

**Clases del diseño**

Una clase del diseño es una abstracción sin costuras de una clase o construcción similar en la implementación del sistema. Esta abstracción es sin costuras en el siguiente sentido:

- El lenguaje utilizado para especificar una clase de diseño es lo mismo que el lenguaje de programación. Consecuentemente las operaciones, parámetros, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido.

- La visibilidad de los atributos y las operaciones de una clase de diseño se especifica con frecuencia.

- Las relaciones de aquellas clases de diseño implicadas con otras clases, a menudo tienen un significado directo cuando la clase es implementada.

Los métodos tienen correspondencia directa con el correspondiente método en la implementación de las clases.

Diagrama de clases del diseño por casos de uso

**Ver Anexos Figuras 30-32**

### 2.7.1.2 Diagramas de Interacción

La secuencia de acciones en un caso de uso comienza cuando un actor invoca el caso de uso mediante el envío de algún tipo de mensaje al sistema. En el interior del sistema se tendrá algún objeto de diseño que recibe el mensaje del actor. Después el objeto de diseño llama a algún otro objeto, y de esta manera los objetos implicados interactúan para realizar y llevar a cabo el caso de uso. En el diseño es preferible representar esto con diagramas de secuencia ya que el centro de atención principal es encontrar secuencias de interacciones detalladas y ordenadas en el tiempo.

Diagramas de secuencia del diseño **Ver índice de figuras 20-23**

## *2.8 Patrones empleados en la solución*

En la solución se emplearon los siguientes patrones de diseño

**Factory Method** (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.

**Singleton:** El Singleton asegura que una clase tenga solo una instancia y provee un punto global de acceso a esta. Es importante para algunas clases tener exactamente una instancia y una vía de lograr esto es hacer responsable a la clase de seguir el rastro de su única instancia, y garantizar que otra no sea creada.

**Facade** (*Fachada*): El patrón Fachada sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.

**Bajo Acoplamiento:**

Este es un principio que se debe recordar siempre que se vaya a diseñar algo. Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también reutilizables, que acrecientan la oportunidad de una mayor

productividad.

**Alta cohesión:**

Este es también un principio que se debe tener en cuenta en todas las decisiones de diseño. Grady Booch señala que se da una alta cohesión cuando los elementos de un componente, una clase por ejemplo, "colaboran para producir algún comportamiento bien definido". Una clase de alta cohesión posee un número relativamente pequeño de responsabilidades, con una importante funcionalidad relacionada y poco trabajo por hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Este patrón mejora la calidad y facilidad con que se puede entender el diseño, se genera un bajo acoplamiento y permite fomentar la reutilización.

**Experto:**

Asignar una responsabilidad al experto de información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Expresa simplemente la intuición de que los objetos hacen cosas relacionadas con la información que poseen. Este patrón permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento. Este patrón propicia además que el comportamiento se distribuya entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas, o sea que además brinda soporte a una alta cohesión.

**Creador:**

Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Este patrón brinda soporte a un bajo acoplamiento.

## *2.9 Conclusiones Parciales*

Las técnicas de requisitos empleadas permitió determinar las funcionalidades principales del subsistema a trabajar, de estas funcionalidades especificadas se obtuvo los casos de uso, los cuales junto con las características del sistema permitió la realización del análisis y el diseño del subsistema Configuración de Datos, generándose los diagramas de clases del análisis y del diseño, así como los de colaboración y secuencia respectivamente. Además se justificó el uso de los patrones de diseño empleados como parte de la propuesta de solución, así como las características de ellos.

# CAPÍTULO 3: VALIDACIÓN DE LOS RESULTADOS.

## *3.1- Introducción*

La medición es el proceso por el que se asignan números o símbolos a los atributos de las entidades del mundo real, de tal manera que las definan de acuerdo con unas reglas claramente definidas... (Fenton, 1991)

La medición es fundamental para cualquier disciplina de ingeniería, y la ingeniería del software no es una excepción. La medición nos permite tener una visión más profunda, proporcionando un mecanismo para la evaluación objetiva. (Pressman, 2005)

En el presente capítulo se realiza un análisis de los resultados, tomando como principal basamento las métricas para determinar la calidad de la especificación de los requisitos, del DCUS y del diseño.

## *3.2- Métricas*

El IEEE Standard Glossary of Software Engineering Terms [IEEE93] define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.
Una medida proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto.

La medición es el acto de determinar una medida.
Hay cuatro razones para medir los procesos del software, los productos y los recursos:

- Caracterizar
- Evaluar
- Predecir
- Mejorar

Caracterizamos para comprender mejor los procesos, los productos, los recursos y los entornos y para establecer las líneas base para las comparaciones con evaluaciones futuras. Evaluamos para determinar el estado con respecto al diseño. También evaluamos para valorar la consecución de los objetivos de calidad y para evaluar el impacto de la tecnología y las mejoras del proceso. Predecimos para poder planificar. Hacemos esto porque queremos establecer objetivos alcanzables para el coste, planificación, y calidad -de manera que se puedan aplicar los recursos apropiados. Medimos para mejorar, la medición nos permite recoger la información cuantitativa que nos ayuda a identificar obstáculos, problemas de raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el rendimiento del proceso. (Pressman, 2005)

### 3.2.1- Métricas de la calidad de la especificación

Davis y sus colegas (Pressman, 2005) proponen una lista de características que pueden emplearse para valorar la calidad del modelo de análisis y la correspondiente especificación de requisitos: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación, exactitud y capacidad de reutilización. Aunque muchas de estas características parecen de naturaleza cualitativa, Davis sugiere que todas pueden representarse usando una o más métricas.

Asumimos que hay $n_r$ requisitos en una especificación, tal como:

$$n_r = n_f + n_{nf}$$

Donde: $n_f$ es el número de requisitos funcionales.

$n_{nf}$ es el número de requisitos no funcionales.

**Especificidad**

Para determinar la especificidad (ausencia de ambigüedad) de los requisitos. Davis sugiere una métrica basada en la consistencia de la interpretación de los revisores para cada requisito:

$$Q_i = {}^{n_{ui}}/_{n_r}$$

Donde: $n_{ui}$ es el número de requisitos para los que todos los revisores tuvieron

interpretaciones idénticas.

Cuanto más cerca este de 1 el valor de Q, menor será la ambigüedad de la especificación.

Para evaluar la métrica de la especificidad de los requisitos se realizaron dos revisiones por varios revisores. El objetivo principal de estas revisiones era obtener los requisitos con el mayor grado de claridad posible y sin ambigüedades.

**Resultados:**

$n_r = n_f + n_{nf}$

$n_r = 10 + 15$

$n_r = 25$

**Primera Revisión**

$Q_i = {}^{n_{ui}}/_{n_r}$

$Q_i = {}^{20}/_{25}$

$Q_i = 0.8$

**Segunda Revisión**

$Q_i = {}^{n_{ui}}/_{n_r}$

$Q_i = {}^{25}/_{25}$

$Q_i = 1.0$

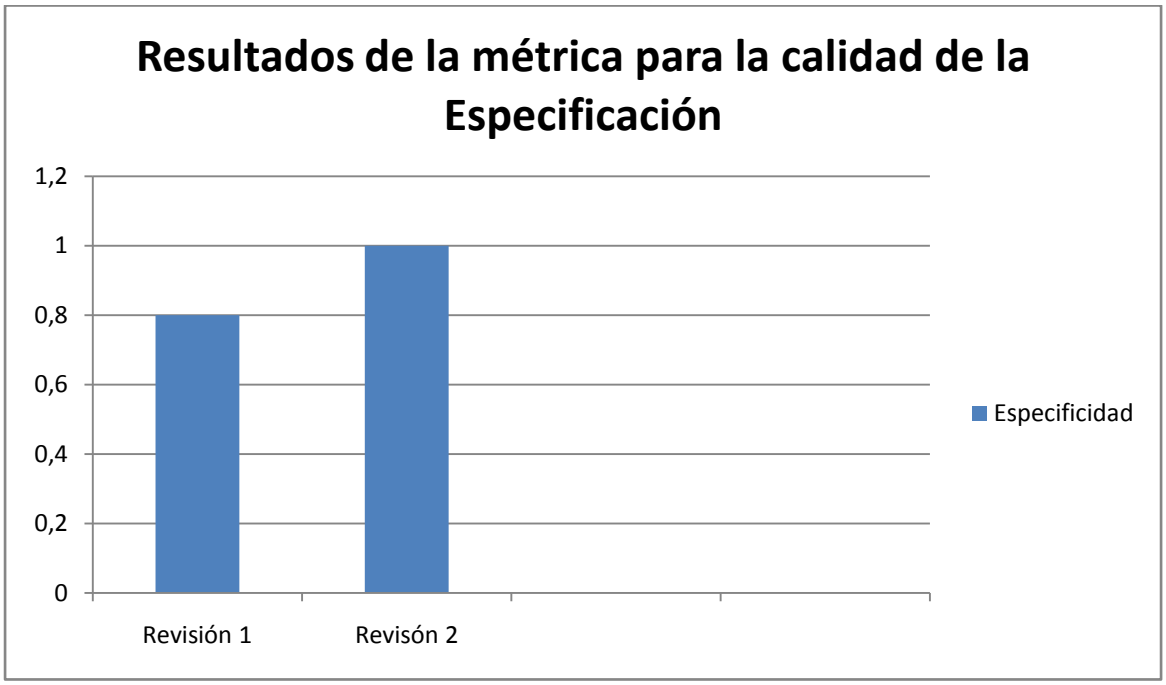



**Figura 3.1** Resultados de la Especificidad

Luego de realizada la primera revisión se detectaron algunos requisitos con problemas de redacción, que dificultaban la interpretación de los revisores. Se realizaron los cambios necesarios y para la segunda revisión todos los revisores tuvieron la misma interpretación en 10 requisitos de un total de 10, lo que significa que todos los requisitos están especificados de forma clara y sin ambigüedades.

### 3.2.2- Métricas de Casos de uso

| Métricas principales | Rango habitual | Descripción |
|---|---|---|
| NOS | [4,9] | Números de Pasos |
| NOAS/NOS | [30%,60%] | Proporción de pasos de actor |
| NOSS/NOS | [40%, 80%] | Proporción de pasos de sistema |
| NOUS/NOS | [0%,35%] | Proporción de pasos de casos de uso |
| CC | [1,5] | Complejidad Ciclomática (NOCS+NOE+1) |

**Tabla 3.1 Métricas principales y rango habitual**

**Métrica NOAS /NOS**

**NOAS:** Number of Actor Steps- Número de pasos del actor

**NOS:** Number of Steps- Número de pasos

Esta heurística se basa en la idea de que un caso de uso sirve para expresar una interacción actor–sistema. Por ello, el número de pasos de actor y el de pasos en general deben estar en torno al 50%, considerando también la posibilidad de que existan pasos de inclusión o extensión en los que se realice otro caso de uso.

Las situaciones que llevan a esta métrica fuera del rango habitual son:

- El hecho de obviar la participación del sistema, por lo que el caso de uso resulta incompleto. Es la situación más habitual.
- El hecho de haber desglosado demasiado las acciones de un actor determinado. En este caso aparecen varios pasos seguidos, del mismo actor, lo cual podría haberse evitado uniéndolos en uno solo, separando las acciones por comas en el texto del paso.

El rango habitual de esta métrica es [30%,60%]. Un valor alto de la misma puede estar condicionado por el hecho de que el caso de uso no incluye todo lo que debe hacer el sistema para alcanzar el objetivo de este o demasiado desglose de los pasos del actor. Un valor bajo de la misma puede estar condicionado por el hecho de que no incluye todo lo que debe hacer el actor para alcanzar el objetivo del caso de uso, demasiado desglose de los pasos del sistema. (Bernárdez, y otros, 2004)

| Nombre del CU | NOAS | NOS | Valor de la métrica |
|---|---|---|---|
| **Gestionar Estructura de la Base de Datos** | 10 | 21 | 47% |
| **Autenticarse** | 8 | 14 | 57% |
| **Mostrar Historial** | 3 | 6 | 50% |
| **Ver Conexión Actual** | 2 | 4 | 50% |

**Tabla 3.2** Aplicación de la métrica NOAS/NOS al DCUS

Esta métrica fue aplicada a cada uno de los casos de uso del sistema, como se muestra en la tabla anterior, para cada uno de los casos de uso el valor de esta métrica está en el rango habitual, con valores bastante cercanos al 50%, lo que implica que los mismos están bastante completos pues no se obvian participaciones del actor o del sistema, o sea, que se incluye todo lo que debe hacer tanto el actor como el sistema, para lograr el objetivo del caso de uso; y los pasos del actor y del sistema tienen un desglose adecuado.

Se aplicaron además un conjunto de métricas orientadas a objetos para evaluar las siguientes propiedades de calidad: **consistencia, correctitud, completitud y complejidad.**

**Completitud:** Grado en que se ha logrado detallar todos los casos de uso relevantes.
**Consistencia:** Grado en que los casos de uso del sistema describen las interacciones adecuadas entre el usuario y el sistema.
**Correctitud:** Grado en que las interacciones actor / sistema soportan adecuadamente el proceso del negocio.
**Complejidad:** Grado de claridad en la presentación de los elementos que describen el contexto y la claridad del sistema.

Estas métricas fueron aplicadas en dos revisiones realizadas:

**Primera Revisión**

| Factor | Métricas Asociadas | Valor |
|---|---|---|
| Factor Completitud | | |
| Factor 1. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/ modificar o consultar información? | Métrica 1. Número de roles relevantes omitidos. | Cantidad de roles relevantes definidos: 1<br>Número de roles relevantes omitidos: 0 Representa un 0%. |
| Factor 2. ¿Se presenta una descripción resumida de todos los conceptos del dominio? | Métrica 2. Número de conceptos del dominio que no presenta una descripción resumida | Número de conceptos del dominio: 17<br>Número de conceptos que no presentan una descripción detallada: 0<br>Representa un 0% |
| Factor 3. ¿Están definidos todos los requisitos que | Métrica 3. Número de requisitos omitidos por caso | Cantidad de requisitos: 14<br>Número de requisitos omitidos por caso de uso: 0<br>Representa un 0% |

| | | |
|---|---|---|
| justifican la funcionalidad del caso de uso? | de uso.<br><br>Métrica 4. Número de casos de uso que tienen requisitos omitidos. | Número de casos de uso: 8<br>Cantidad de casos de uso que tienen requisitos omitidos: 0<br><br>Representa un 0% |
| Factor 4. ¿Existen requisitos que no han sido considerados en algún caso de uso? | Métrica 5. Número de requisitos que no son considerados en ningún caso de uso. | Número de requisitos: 14<br>Número de requisitos que no son considerados en ningún caso de uso 0<br>Representa un 0% |
| Factor 5. ¿Se describen las condiciones de excepción relevantes que debe contemplar cada flujo de eventos? | Métrica 6. Número de casos de uso que no describen condiciones de excepción relevantes. | Total de Casos de uso: 8<br>Cantidad de casos de uso que no describen condiciones de excepción relevantes: 3<br><br>Representa un 37.5% |
| Factor 6. ¿Todos los casos de uso han sido clasificados de acuerdo a su relevancia en (crítico, secundario, auxiliar, opcional)? | Métrica 7. Número casos de que no han sido clasificados | Total de casos de uso: 8<br>Cantidad de casos de uso que no han sido clasificados: 0%<br>Representa un 0% |
| Factor 7. ¿Se presenta una descripción resumida de todos los casos de uso? | Métrica 8. Número de casos de uso que no tiene descripción resumida. | Total de casos de uso: 8<br>Cantidad que no tienen una descripción resumida: 0<br>Representa un 0% |
| Factor 8. ¿Se presenta una descripción detallada (descripción extendida esencial) de todos los casos de uso? | Métrica 9: Número de casos de uso que no poseen una descripción extendida. | Total de casos de uso: 8<br>Cantidad que no tienen una descripción extendida: 0<br>Representa un 0% |
| | | **95.34%** |
| **Factor Consistencia** | | |
| Factor 9. ¿Representa el caso de uso una interacción observable por un actor? | Métrica 10. Número de casos de uso que no representan una interacción observable por un actor | Total de casos de uso: 8<br>Cantidad que no representan una interacción observable por un actor: 0<br>Representa un 0% |
| Factor10. ¿Está adecuadamente redactado (en el lenguaje del usuario) el | Métrica 11. Cantidad de casos de uso cuyo flujo de eventos está redactado en el lenguaje | Total de casos de uso: 8<br>Cantidad que no tienen el flujo de eventos redactado en el |

| | | |
|---|---|---|
| flujo de eventos? | del usuario | lenguaje del usuario: 0<br>Representa un 0% |
| Factor 11. ¿Existe una adecuada separación entre el flujo básico de eventos y los flujos alternos y/o flujos subordinados? | Métrica 12. Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos | Total de casos de uso: 8<br>Cantidad de casos de uso complejos que no tienen una separación del flujo básico y de flujos alternos: 0<br>Representa un 0% |
| Factor 12. ¿El nombre dado a los casos de uso es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario? | Métrica 13: Número de casos de uso que tienen un nombre incorrecto | Total de casos de uso: 8<br>Cantidad que tienen un nombre incorrecto: 0<br>Representa un 0% |
| | | **100%** |
| **Factor de Correctitud** | | |
| Factor 13. ¿Representa el caso de uso requisitos comprensibles por el usuario? | Métrica14. Grado en que los requisitos representados por el caso de uso son comprensibles por el usuario<br><br>Métrica 15. Número de casos de uso en que los requisitos representados no son comprensibles por el usuario | Total de requisitos: 14<br>Cantidad de requisitos que no son comprensibles por el usuario: 0<br>Representa: 0%<br><br>Total de casos de Uso: 8<br>Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 0<br>Representa: 0% |
| Factor 14. ¿Las interacciones definidas describen la funcionalidad requerida del sistema? | Métrica 16. Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema | Total de casos de Uso: 8<br>Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema : 3<br><br>Representa: 37.5% |
| Factor 15. Las interacciones definidas introducen mejoras al proceso actual. | Métrica 17. Número de casos de uso que deben ser modificados para mejorar el proceso actual | Total de casos de Uso: 8<br>Número de casos de uso que deben ser modificados para mejorar el proceso actual: 3<br><br>Representa: 37.5% |
| | | **92.89%** |

| | | |
|---|---|---|
| **Factor de Complejidad** | | |
| Factor 16 ¿Los elementos dentro del diagrama están adecuadamente ubicados de manera que facilitan su | Métrica 19. Número de elementos del diagrama que requieren reubicación | Total de casos de Uso: 8.<br>Número de casos de uso que requieren reubicación de manera que faciliten su |

| | | |
|---|---|---|
| interpretación? | | interpretación:1<br>Representa: 12.5 % |
| | | **97.6%** |

**Segunda Revisión**

| Factor | Métricas Asociadas | Valor |
|---|---|---|
| Factor Completitud | | |
| Factor 1. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/ modificar o consultar información? | Métrica 1. Número de roles relevantes omitidos. | Cantidad de roles relevantes definidos: 1<br>Número de roles relevantes omitidos: 0<br>Representa un 0%. |
| Factor 2. ¿Se presenta una descripción resumida de todos los conceptos del dominio? | Métrica 2. Número de conceptos del dominio que no presenta una descripción resumida | Número de conceptos del dominio: 17<br>Número de conceptos que no presentan una descripción detallada: 0<br>Representa un 0% |
| Factor 3. ¿Están definidos todos los requisitos que justifican la funcionalidad del caso de uso? | Métrica 3. Número de requisitos omitidos por caso de uso.<br><br>Métrica 4. Número de casos de uso que tienen requisitos omitidos. | Cantidad de requisitos: 10<br>Número de requisitos omitidos por caso de uso: 0<br><br>Representa un 0%<br><br>Número de casos de uso: 4<br>Cantidad de casos de uso que tienen requisitos omitidos: 0<br><br>Representa un 0% |
| Métrica 4. Número de casos de uso que tienen requisitos omitidos. | Métrica 5. Número de requisitos que no son considerados en ningún caso de uso. | Número de requisitos:10<br>Número de requisitos que no son considerados en ningún caso de uso 0<br>Representa un 0% |
| Factor 5. ¿Se describen las condiciones de excepción relevantes que debe contemplar cada flujo de eventos? | Métrica 6. Número de casos de uso que no describen condiciones de excepción relevantes. | Total de Casos de uso:4<br>Cantidad de casos de uso que no describen condiciones de excepción relevantes: 0 |

| | | |
|---|---|---|
| | | Representa un 0% |
| Factor 6. ¿Todos los casos de uso han sido clasificados de acuerdo a su relevancia en (crítico, secundario, auxiliar, opcional)? | Métrica 7. Número casos de que no han sido clasificados | Total de casos de uso: 4<br>Cantidad de casos de uso que no han sido clasificados: 0%<br><br>Representa un 0% |
| Factor 7. ¿Se presenta una descripción resumida de todos los casos de uso? | Métrica 8. Número de casos de uso que no tiene descripción resumida. | Total de casos de uso: 4<br>Cantidad que no tienen una descripción resumida: 0<br><br>Representa un 0% |
| Factor 8. ¿Se presenta una descripción detallada (descripción extendida esencial) de todos los casos de uso? | Métrica 9: Número de casos de uso que no poseen una descripción extendida. | Total de casos de uso: 6<br>Cantidad que no tienen una descripción extendida: 0<br><br>Representa un 0% |
| | | **100%** |
| **Factor Consistencia** | | |
| Factor 9. ¿Representa el caso de uso una interacción observable por un actor? | Métrica 10. Número de casos de uso que no representan una interacción observable por un actor | Total de casos de uso: 4<br>Cantidad que no representan una interacción observable por un actor: 0<br><br>Representa un 0% |
| Factor10. ¿Está adecuadamente redactado (en el lenguaje del usuario) el flujo de eventos? | Métrica 11. Cantidad de casos de uso cuyo flujo de eventos está redactado en el lenguaje del usuario | Total de casos de uso: 4<br>Cantidad que no tienen el flujo de eventos redactado en el lenguaje del usuario: 0<br>Representa un 0% |
| Factor 11. ¿Existe una adecuada separación entre el flujo básico de eventos y los flujos alternos y/o flujos subordinados? | Métrica 12. Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos | Total de casos de uso: 4<br>Cantidad de casos de uso complejos que no tienen una separación del flujo básico y de flujos alternos: 0<br><br>Representa un 0% |
| Factor 12. ¿El nombre dado a los casos de uso es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario? | Métrica 13: Número de casos de uso que tienen un nombre incorrecto | Total de casos de uso: 4<br>Cantidad que tienen un nombre incorrecto: 0<br>Representa un 0% |

| | | |
|---|---|---|
| | | **100%** |
| **Factor de Correctitud** | | |
| Factor 13. ¿Representa el caso de uso requisitos comprensibles por el usuario? | Métrica14. Grado en que los requisitos representados por el caso de uso son comprensibles por el usuario<br><br>Métrica 15. Número de casos de uso en que los requisitos representados no son comprensibles por el usuario | Total de requisitos: 10<br>Cantidad de requisitos que no son comprensibles por el usuario: 0<br>Representa: 0%<br><br>Total de casos de Uso: 4<br>Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 0<br>Representa: 0% |
| Factor 14. ¿Las interacciones definidas describen la funcionalidad requerida del sistema? | Métrica 16. Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema | Total de casos de Uso: 4<br>Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema : 0<br>Representa: 0% |
| Factor 15. Las interacciones definidas introducen mejoras al proceso actual. | Métrica 17. Número de casos de uso que deben ser modificados para mejorar el proceso actual | Total de casos de Uso: 4<br>Número de casos de uso que deben ser modificados para mejorar el proceso actual: 1<br>Representa: 25.0 % |
| | | **98.7%** |
| **Factor de Complejidad** | | |
| Factor 16¿Los elementos dentro del diagrama están adecuadamente ubicados de manera que facilitan su interpretación? | Métrica 19. Número de elementos del diagrama que requieren reubicación | Total de casos de Uso: 4<br>Número de casos de uso que requieren reubicación de manera que faciliten su interpretación.<br>Representa: 0% |
| | | 100% |

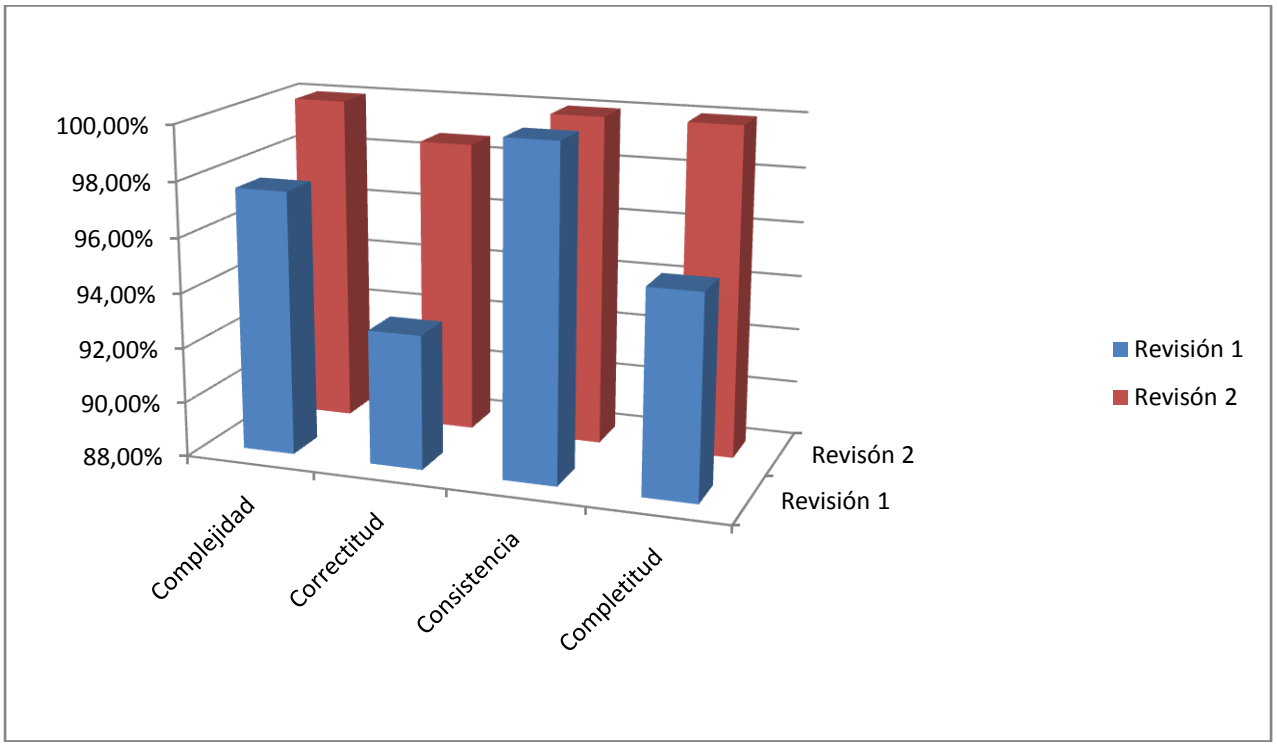



**Figura 3.2** Comportamiento de los principales factores de calidad durante las dos revisiones realizadas al DCUS

El gráfico de la figura permite observar las mejorías en el DCUS de la segunda revisión con respecto a la primera, pues luego de haber aplicado las métricas en la primera revisión se obtuvieron los siguientes resultados:

- El factor completitud se vio afectado pues había casos de uso que no incluían condiciones de excepción relevantes.

- El factor correctitud se vio afectado pues había casos de uso casos de uso que debían ser modificados para adecuarlos a la funcionalidad del sistema.

- El factor complejidad se vio afectado pues había casos de uso que requerían reubicación en el DCUS para que facilitaran su interpretación.

Teniendo en cuenta los resultados de la primera revisión se llevaron a cabo algunos cambios en el DCUS, como parte de los mismos fue necesario eliminar casos de uso, así como redefinir algunos ya existentes y agregar nuevas funcionalidades al sistema, lo mismo que con los

requisitos funcionales los cuales algunos fueron modificados, otros eliminados y otros que aparecieron nuevos. Después de aplicar estos cambios se llevó a cabo una segunda revisión en la cual solo se vio afectado el factor completitud debido a que en uno de los casos de uso que fue agregado se omitió una condición de excepción relevante. Los otros factores de calidad no se vieron afectados pues de manera general los casos de uso están adecuadamente redactados, los mismo representan una interacción observable por un actor, existe una adecuada separación entre el flujo básico y los flujos alternos; y los nombres de los casos de uso son correctos, pues en todos los casos el mismo es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario.

### 3.2.3- Métricas para el Modelo de Diseño

**Métricas Propuestas por Lorenz y Kidd**

- **Tamaño de clase (TC)**

Las métricas orientadas a tamaños para una clase 00 se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema 00 en su totalidad.

El tamaño general de una clase puede medirse determinando las siguientes medidas:

✓ Total de operaciones (tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.

✓ El número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

Un TC grande afecta los indicadores de calidad definidos para esta métrica por los especialistas:

**Reutilización:** reduce la reutilización de la clase.
**Implementación:** complica la implementación.
**Responsabilidad:** la clase debe tener bastante responsabilidad.

Las medidas o umbrales para los parámetros de calidad han sido una polémica a nivel mundial en el diseño de sistemas. Algunos especialistas plantean umbrales para estas métricas según se muestra en la tabla, estos fueron los aplicados en el diseño de este sistema.

| No de Operaciones y/o Atributos | |
|---|---|
| **TC** | **Umbral** |
| Pequeño | <=20 |
| Medio | >20 y <=30 |
| Grande | >30 |

**Tabla 3.3** Umbrales para TC.

Durante el diseño se obtuvieron un total de **30 clases**, de ellas **11 Interfaces**, **3 Controladoras** y **8 Entidades**

**Resultado**: Esta métrica fue aplicada en la capa de presentación y en la capa acceso a datos. En la capa de presentación hay un total de 11 clases, y de acuerdo con los umbrales que se presentan en la tabla todas las clases se pueden considerar de tamaño pequeño. La capa de acceso a datos tiene un total de 8 clases, todas las clases de acuerdo con los umbrales de la tabla se pueden considerar pequeñas. Estos valores demuestran que los indicadores de calidad reutilización, implementación y responsabilidad no se ven afectados.

## *3.3 Conclusiones parciales*

En este capítulo se aplicaron métricas para evaluar el DCUS, la especificación de los requisitos, así como el diseño del sistema, por lo que se puede concluir que:

- Tanto la especificación de los requisitos como el DCUS fueron realizados con calidad. Existe una trazabilidad entre los mismos, pues todos los requisitos están representados en algún caso de uso. Los casos de uso están descritos correctamente, y los elementos dentro del diagrama están adecuadamente distribuidos.

- El sistema propuesto no presenta una alta complejidad permitiendo que las pruebas no sean complejas y que el resto de los módulos puedan ser integrados sin muchas dificultades.
- El sistema presenta un bajo acoplamiento pues la profundidad de los niveles de herencia están acorde con los umbrales.

# Conclusiones generales

- Se obtuvo el modelo del dominio del sistema que permitió definir y relacionar todos los conceptos del mismo.
- Se especificaron y validaron los requisitos funcionales y no funcionales del subsistema.
- Se realizó el análisis del subsistema Configuración de Datos.
- Se diseñó de manera flexible y robusta el subsistema Configuración de Datos.
- Se obtuvieron y evaluaron artefactos necesarios según la metodología seleccionada (RUP), para que se le pueda dar continuidad al proceso de desarrollo.
- Se realizó la validación de los resultados para demostrar la factibilidad del trabajo realizado.

# Recomendaciones

- Que se continúe con el resto de los flujos de trabajo que propone la metodología RUP (implementación y prueba).

- Una vez terminado el producto, sea puesto en explotación por los proyectos productivos de la Facultad.

# Bibliografía

**Bauer, F. L. (1972).** *Software Engineering Information Processing.* Amsterdam: North Holland Publishing Co.

**Becerra González, O., & Durand Martínez, L. (2007-2008).** *Análisis y Diseño de un Sistema para la Generación de Reportes.* Ciudad Habana.

**Boggs, W., & Boggs, M. (2002).** *UML with Rational Rose.*

**Bohem, B. W. (1976).** *Software Engineering.*

**Brown, A. W. (1996).** *Component-Based Software Engineering. s.l.: IEEE Computer Society.* Los Alamitos, CA, 1996.

**Canós, J. H., Letelier, P., & Penadés, M. C.** *Metodologías Ágiles en el Desarrollo de Software.* Valencia.

**Corporation**., R. S. *Rational Unified Process 1.0. s.l. : Corporation, Rational Software, 2003.*

***Electronic Computer Glossary*. (1995).**

**Fernández Sánchez, L. (2007, 3 15).** *Procedimiento para el desarrollo del proceso de ingeniería de requisitos en un proyecto software (PROCIR).* Retrieved 2 2009, from Procedimiento para el desarrollo del proceso de ingeniería de requisitos en un proyecto software (PROCIR).: http://www.informaticahabana.com/evento_virtual.

**Ferrer Grau, X. (2005, 2 15).** *Tesis de Doctorado Marco de Integración de la usabilidad en el Proceso de Desarrollo de Software. Universidad Politécnica de Valencia, Facultad de Informática*. Retrieved 2 2009, from Tesis de Doctorado Marco de Integración de la usabilidad en el Proceso de Desarrollo de Software. Universidad Politécnica de Valencia, Facultad de Informática: http://oa.upm.es/440/01/XAVIER_FERRE_GRAU.PDF

**García Ávila, L. (2007, 3 15).** *Procedimiento para el desarrollo del proceso de ingeniería de requisitos en un proyecto software (PROCIR).* Retrieved 2 2009, from Procedimiento para el desarrollo del proceso de ingeniería de requisitos en un proyecto software (PROCIR).: http://www.informaticahabana.com/evento_virtual.

**García Rubio, F. O., & Bravo Santos, C. (2007, 12 11).** *Metodologías de Desarrollo de Softaware.* Retrieved 2 2009, from Metodologías de Desarrollo de Softaware.: http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/tema9_2xh.pdf.

**Gerardo Fernándes, E. (2002).** *Introducción a Extreme Programming.* Retrieved from Introducción a Extreme Programming.: http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

**IEEE. (1993).** *IEEE Standars Collection: Software Engineering.*

**Jacobson. (2004).** *El Proceso Unificado de Desarrollo.* La Habana: Félix Varela, 2004.

**Jacobson, Ivar, Rumbaugh, Booch, J. y., & Booch, J. y. (2004).** *El Proceso Unificado de Desarrollo. La Habana.* Félix Varela, 2004.

**James, Jacobson, Ivar, Grady, & Rumbaugh. (2000).** *El Proceso Unificado de Desarrollo de Software. s.l.* Addison Wesley.

**Jarzabek, S., & Huang, R. (1998).** *The case for user-centered case tools. s.l. : Communication of the ACM.* 1998. nro 8.

**José Escalona, M., & Koch, N. (2002, 12).** *Ingeniería de Requisitos en Aplicaciones para la WEb. Un estudio comparativo. Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla.* Retrieved 2009, from Ingeniería de Requisitos en Aplicaciones para la WEb. Un estudio comparativo. Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla.: http://www.lsi.us.es/docs/informes/LSI-2002-4.pdf.

**Joyanes Aguilar, D. L. (2000).** *Prólogo a la edición en español del Lenguaje Unificado de Modelado.* Madrid: s.n., Mayo 2000.

**Kendall, K., & Kendall, J. (1997).** *Análisis y Diseño de Sistemas.* México: Tercera Edición.

**Kruchten, P**. *A Rational Development Process. s.l.* STSC, Hill AFB, UT.

**Kruchten, P. (2000).** *The Rational Unified Process: An Introduction. s.l.* Addison Wesley, 2000.

**Larman, C. (1999).** *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* México: Prentice Hall, 1999.

**López Barrio, C. (2005).** *Metodología de desarrollo: Programación Extrema . s.l.: Programa de Doctorado Ingeniería de Sistemas Electrónicos para Entornos Inteligentes.*

**Marqués, J. M. (2005, 9 16).** *Ingeniería del software.* Retrieved 2 2009, from Ingeniería del software.: http://www.infor.uva.es/~jmmc/ingsoft/isprograma.html.

**Pressman, R. S. (1998).** *Ingeniería de Software. Un enfoque práctico.* Madrid: Mc Graw-Hill Interamericana de España S.A.

**Rumbaugh, & Grady. (2000).** *El Proceso Unificado de Desarrollo de Software. s.l.* Addison Wesley.

**Sánchez, P., Maglema, R., & Vázquez Baños, Y. (2007).** *Sistema Generador de Mapas Temáticos y Gráficos Estadísticos.* La Habana: s.n., 2007.

**Schmuller, J. (2000).** *Aprendiendo UML en 24 horas.* México: Pearson Educación, 2000.

**Tejera Hernández, D. C., & Sánchez Echevarría, L. B. (2007).** *Ingeniería de Requisitos para el desarrollo del Sistema de Inventario Almacén (SIGIA) Módulo Nomencladores.* s.n., 2007: La Habana.

***Teleformación*.** (n.d.). Retrieved 1 2009, from http://teleformacion.uci.cu

**Zavala, R. (2000).** *Diseño de un Sistema de Información Geográfica sobre Internet. Tesis de Maestría en Ciencias de la Computación.* México: s.n., 2000.

**Zelkovitz, M. V. (1976).** *Principies of Software Engineering and Design. s.l.* Prentice Hall.

# Anexos

## Anexo 1. Principios del Manifiesto Ágil

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

**Anexo 2. Artefacto Glosario de Términos**

**Archivo de Reporte:** Archivo que contiene la información referente al diseño y el origen de datos de un Reporte. Es creado por el Diseñador y usado por el Visor para visualizar el Reporte. *RPT.*

**Área de Diseño:** Área del Diseñador en la que se visualiza y configura el diseño del Reporte. Está compuesta por una o más *Secciones*.

**Cuadro de Herramientas:** Componente Visual que contiene todas las *Herramientas* que pueden ser usadas para diseñar un Reporte.

**Diseñador:** Hace referencia al *Diseñador de Reportes. Diseñador de Informes.*

**Diseñador de Informes:** Hace referencia al *Diseñador de Reportes*. *Diseñador.*

**Diseñador de Reportes:** Es un Módulo del Sistema que permite al usuario crear o modificar el diseño de un reporte, brindando las funcionalidades requeridas para ello y permitiendo guardarlo en un Archivo de Reporte para su posterior utilización. *Diseñador, Diseñador de Informes.*

**Fuente de Datos:** Lugar de procedencia de los datos contenidos en el *Reporte. Origen de Datos.*

**Generador de Reportes:** Herramienta informática que da soporte a los procesos de diseño y presentación de los Reportes de un sistema o aplicación. *Reporteador.*

**Herramienta:** Es un objeto o componente que permite realizar operaciones sobre el Área de Diseño. Con un alto nivel de abstracción, podría considerarse que las herramientas “trabajan” o “dibujan” sobre el Área de Diseño.

**Informe:** *Reporte.*

**Módulo de Comunicación Exterior:** es un Módulo del Sistema que permite a la comunicación e interacción con otros sistemas y Aplicaciones Usuarias.

**Objeto de Diseño:** es un elemento del diseño del reporte. Ej. Cuadro de texto, línea, gráfico, imagen.

**Reporte:** Conjunto de datos relacionados entre sí y con un determinado tema, que son presentados en un formato y estructura específicos. *Informe.* En otro contexto, puede hacer referencia al Archivo de Reporte o RPT.

**Reporteador:** Hace referencia a un *Sistema Generador de Reportes.*

**RPT o .RPT:** *Archivo de Reporte.*

**Sección**: es una agrupación lógica del diseño del reporte.

**Sección Encabezado de Reporte:** El contenido de esta Sección solo se muestra una vez al principio del Reporte.

**Sección Encabezado de Página:** El contenido de esta Sección se muestra al principio de cada página del Reporte, salvo en la primera página donde se muestra después de la Sección Encabezado de Reporte.

**Sección Detalles:** En esta Sección se coloca el contenido repetitivo del Reporte. Ejemplo: si el reporte representa el contenido de una tabla en una Base de Datos, en esta sección deben colocarse los campos de la misma que se deseen mostrar, luego al visualizar el reporte, se generará una línea por cada tupla de la tabla.

**Sección Pie de Reporte:** El contenido de esta Sección se muestra sólo en la última página del Reporte. Su contenido se comienza a mostrar de abajo hacia arriba inmediatamente después del la Sección Pie de Página.

**Sección Pie de Página:** El contenido de esta Sección se muestra al final de cada página del Reporte. Su contenido se comienza a mostrar de abajo hacia arriba.

**Visor:** Hace referencia al *Visor de Reportes.*

**Visor de Reportes:** es un Módulo del Sistema que permite la visualización de un Reporte previamente diseñado, permitiendo además su impresión y exportación a diferentes formatos.
**Principales Eventos:**

**Contiene:** expresa que A está contenido lógicamente en B, Generador de Reportes contiene Diseñador de Informes y Visor de Informes, Diseñador de Informes contiene Área de Diseño y Cuadro de Herramientas, Cuadro de Herramientas contiene Herramientas, el Área de Diseño contiene Sección y la Sección contiene objeto de diseño

**Utiliza:** proceso mediante el usuario utiliza el sistema generador de reportes.

**Crea y modifica:** proceso mediante el cual se crea y modifica el diseño del reporte.

**Carga:** proceso mediante el cual se carga el diseño del reporte.

**Editar reporte:** proceso mediante el cual se modifica un reporte existente.

**Vista Previa:** proceso mediante el cual se obtiene una vista previa del reporte mientras se diseña.

**Genera, Visualiza:** proceso mediante el cual se visualiza el reporte una vez terminado su diseño.

**Exporta:** proceso mediante el cual se exporta el reporte.

**Imprime:** proceso mediante el cual se imprime el reporte.

**Configuración de Datos:** Es un modulo del sistema que le permite al usuario todo el trabajo con la Base de Datos, permitiéndole ver las conexiones, la ubicación de las bases de datos así como las estructuras de las tablas de la Base de Datos.

**Estructura de Datos:** La forma en la que se encuentran los datos en la base de datos es decir las tablas con estos datos seria lo que conformaría la estructura de datos.

**Origen de Datos**: Fuente de Datos. Es un paquete que va a contener la localización física de la Base de Datos, las autenticaciones y los drivers de conexión, es decir la ubicación de la base de datos y lo que se necesita para conectarse a ella.

**Objeto de Conexión:** Va a ser un objeto que le va a brindar al diseñador los datos obtenidos del modulo configuración de datos y que a su ves va a tener la estructura de los datos y el origen de datos.

**Localización física de la BD:** Ubicación real de la BD.

**Anexo 3. Características del sistema**

**Características Generales**

1- Basada en Software Libre.

2- Multiplataforma.

3- Independiente del lenguaje de programación utilizado.

4- Conexión a múltiples bases de datos.

5- Diseñador gráfico del reporte (básico y avanzado).

6- Editor de Expresiones.

7- Funciones básicas (número de página, promedio, suma…).

8- Soporte para imágenes, gráficos, líneas, formas predeterminadas y textos independientes.

9- Exportar a múltiples formatos (PDF, HTML, RTF,…).

10- Configuración de página y Opciones de impresión.

11- Soporte para sub-reportes.

12- Soporte para herencia entre reportes.

13- Vista previa dinámica.

14- Soporte para edición de los reportes de manera dinámica en tiempo de ejecución.

**Soporte de bases de datos**

1- Soporte para múltiples bases de datos.

2- Soporte para consultas parametrizadas y procedimientos almacenados.

3- Soporte para archivos bases de datos (.xsd, .xml…).

4- Soporte para datasets relacionales (dataset con relaciones entre tablas) *(linked dataset).*

5- Diseñador de fuente de datos, exportable a XSD y XML.

**Características de Diseño de Reportes**

1- Número ilimitado de sub-reportes.

2- Cada subreporte puede opcionalmente contener su propio dataset.

3- Número ilimitado de secciones [Encabezado de reporte, encabezado de página, detalles, pie de página y pie de reporte].

4- Número ilimitado de grupos.

5- Funciones básicas de grupos (Promedio, subtotal, gran total, contador, máximo, mínimo,…).

6- Funciones para el trabajo con diferentes tipos de datos:

Funciones de cadena (concatenar (s1, s2), subcadena(s, posIni, posFin),…).

Funciones numéricas (suma(x, y), multiplicación(x, y),…).

7- Utilidades de precisión (regla, cuadrícula, alineamiento automático, unidades de medida (mm, cm, pulgadas,…),…).

8- Facilidades de ampliación de la vista (zoom in, zoom out).

9- Fondo del reporte configurable (gradientes, texturas, tramas, imágenes).

10- Fondo de sección configurable (gradientes, texturas, tramas, imágenes).

11- Soporte para múltiples formatos de imagen (JPG, BMP, GIF).

12- Soporte para gráficos.

13- Soporte para líneas y formas predeterminadas.

14- Soporte para textos independientes configurables (negritas, subrayado, color de letras, color de fondo).

15- Soporte para hipervínculos.

16- Soporte para múltiples formatos de texto (negritas, subrayado, itálica,…).

17- Formatos por defecto configurables para cada tipo de dato.

18- Formato numérico configurable (separador decimal, separador de miles, cifras decimales, redondeo,…).

19- Facilidades de edición (Multiselección, formato a múltiples objetos, copiar, pegar,…).

**Opciones de Impresión**

1- Página configurable (tamaño, posición).

2- Márgenes y sangría.

3- Selección y configuración de impresora.

4- Vista previa dinámica con filosofía WYSIWYG *(What You See Is What You Get).*

**Diseñador gráfico de reportes**

1- Cuenta con dos modos de presentación:

**Modo Básico:** Pensado para usuarios no expertos en informática, como usuarios finales y clientes de aplicaciones.

1- Muestra una interfaz reducida, sencilla y muy amigable.

2- Presenta un conjunto reducido de funcionalidades, tal que satisfaga solo las necesidades básicas de diseño.

**Modo Avanzado:** Pensado para usuarios expertos en informática, como los desarrolladores y personal de soporte de aplicaciones.

1- Muestra una interfaz más cargada y funcional, sin dejar de ser amigable.

2- Presenta el conjunto completo de funcionalidades, tal que puedan ser explotadas al máximo las potencialidades de los reportes.

**Anexo 4. Entrevista aplicada a los desarrolladores de reportes de la universidad.**

Los resultados de esta entrevista contribuirán a desarrollar un estudio sobre el uso de las herramientas para generar reportes usadas en la UCI, y contribuirán a establecer las bases teóricas para el modelado de un sistema de generación de reportes.

1) ¿Cuánto tiempo de experiencia tiene en el diseño de reporte?

2) ¿En qué ambiente de desarrollo trabajan?

3) ¿Qué herramienta usan para generar los reportes?

4) ¿Qué ventajas y desventajas presenta la herramienta seleccionada?

5) ¿Qué harían si necesitaran incluir un nuevo reporte en el sistema después de desplegado?

6) ¿Cree Ud. que un usuario final sea capaz de modelar un reporte de acuerdo a sus propias necesidades usando esa herramienta?

7) ¿Qué otras herramientas para generar reportes conoce o fueron estudiadas?

8) ¿Qué pudiera decir de ellas?

9) ¿Qué opina de las herramientas construidas por estudiantes de nuestra universidad?

10) Algún aporte que pueda hacer al desarrollo de una nueva herramienta para la generación de reportes: ideas, características importantes, sugerencias, documentación, etc

# Figuras

Figura 1 Prototipo Interfaz Usuario, Configuración de Datos


Figura 2 Prototipo Interfaz Usuario CU Gestionar Estructura de la Base de Datos, Sección Nuevo Origen de Datos.

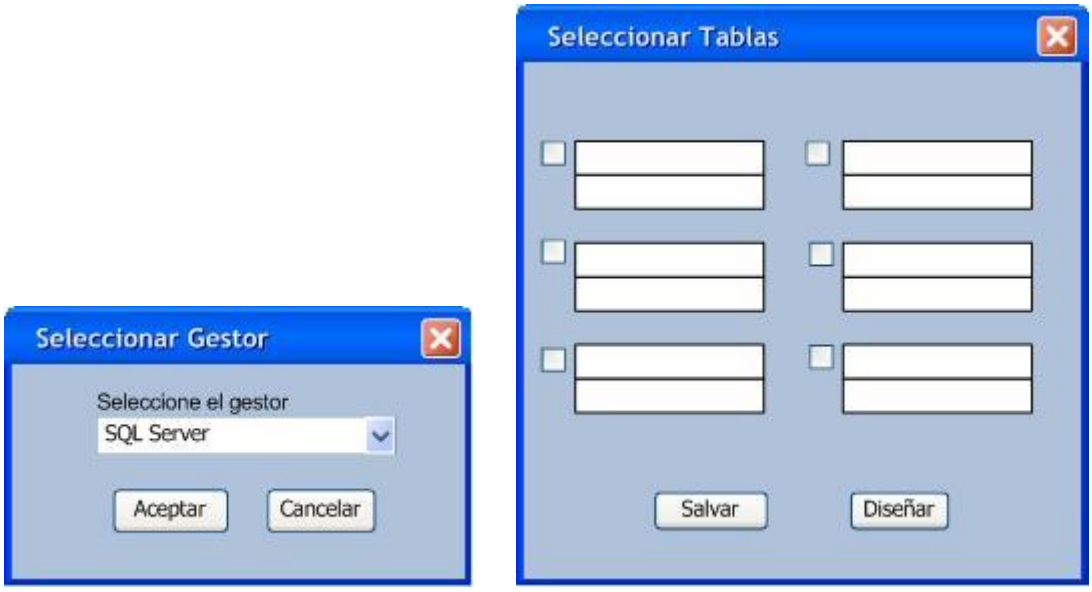


Figura 3 Prototipo Interfaz Usuario CU Gestionar Estructura de la Base de Datos, Sección Gestionar Origen de Datos.


Figura 4 Prototipo Interfaz Usuario CU Autenticarse para Postgres

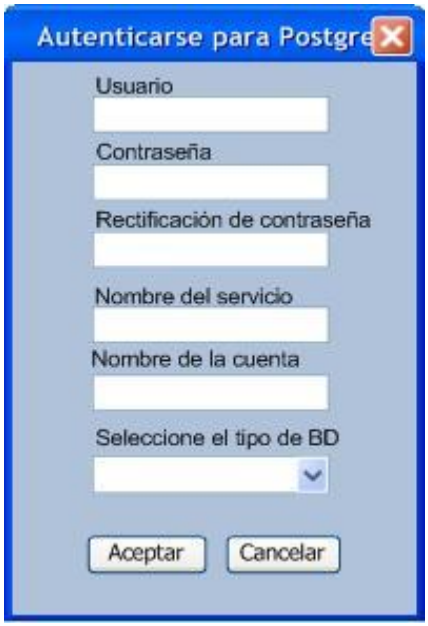


Figura 5 Prototipo Interfaz Usuario CU Autenticarse Oracle

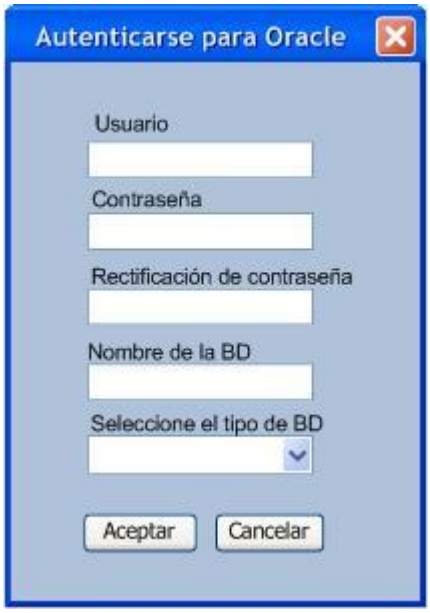



Figura 6 Prototipo Interfaz Usuario CU Autenticarse para MySQL

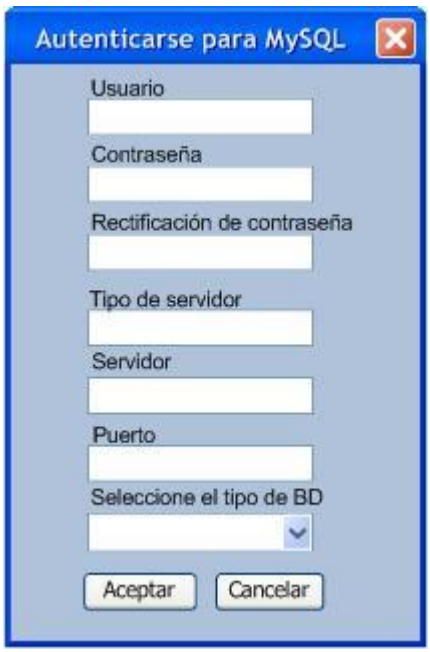


Figura 7 Prototipo Interfaz Usuario CU Autenticarse para Access

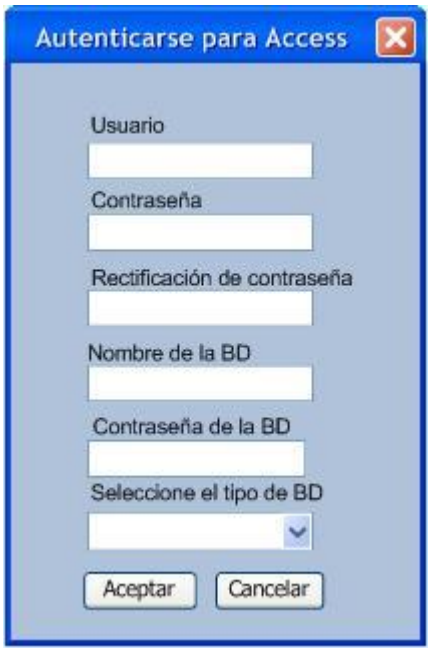



Figura 8 Prototipo Interfaz Usuario CU Autenticarse para SQL Server

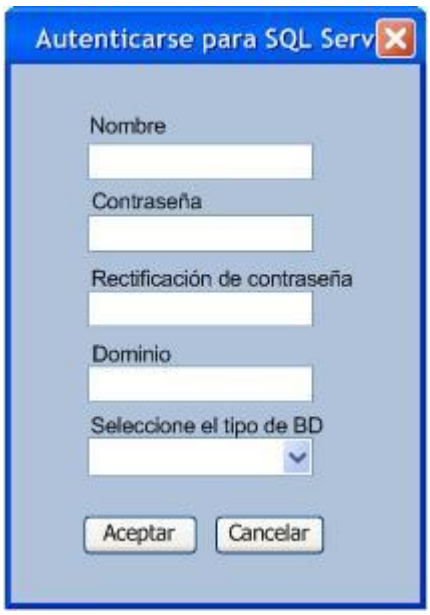


Figura 9 Prototipo Interfaz Usuario CU Mostrar Historial


Figura 10 Prototipo Interfaz Usuario CU Conexión Actual

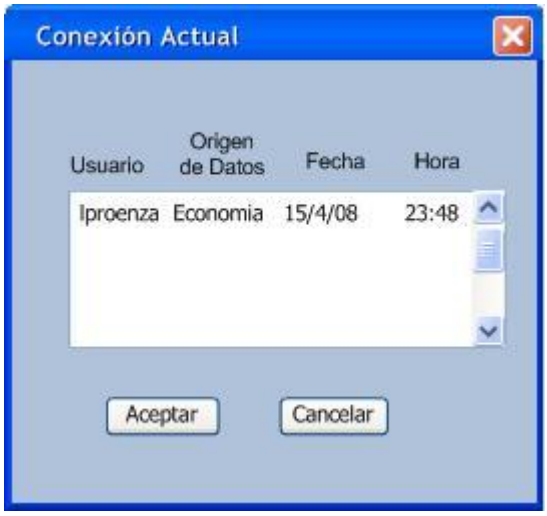


Figura 11 Diagrama de Colaboración CU Gestionar Estructura de la Base de Datos, Sección GestionarOD.

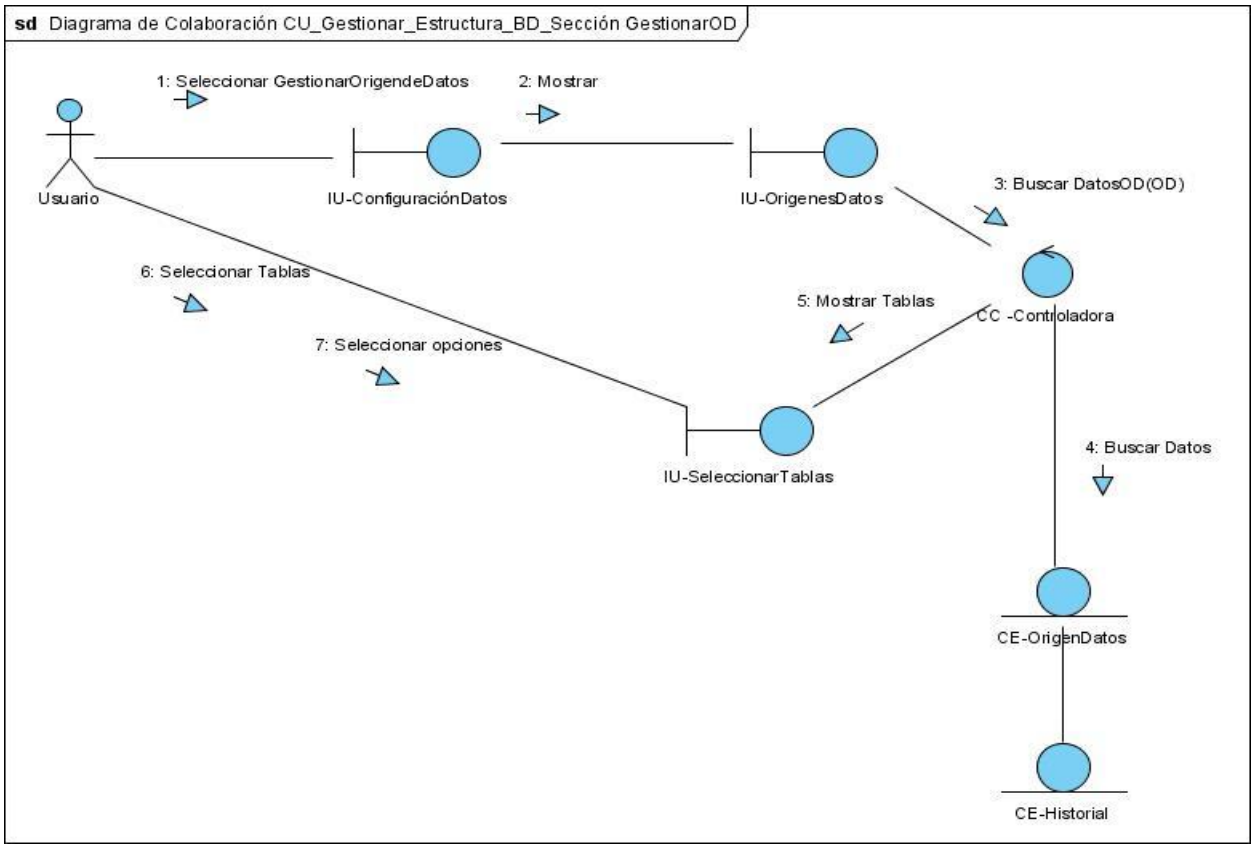



Figura 12 Diagrama de Colaboración CU Gestionar Estructura de la Base de Datos, Sección Nuevo Origen de Datos.

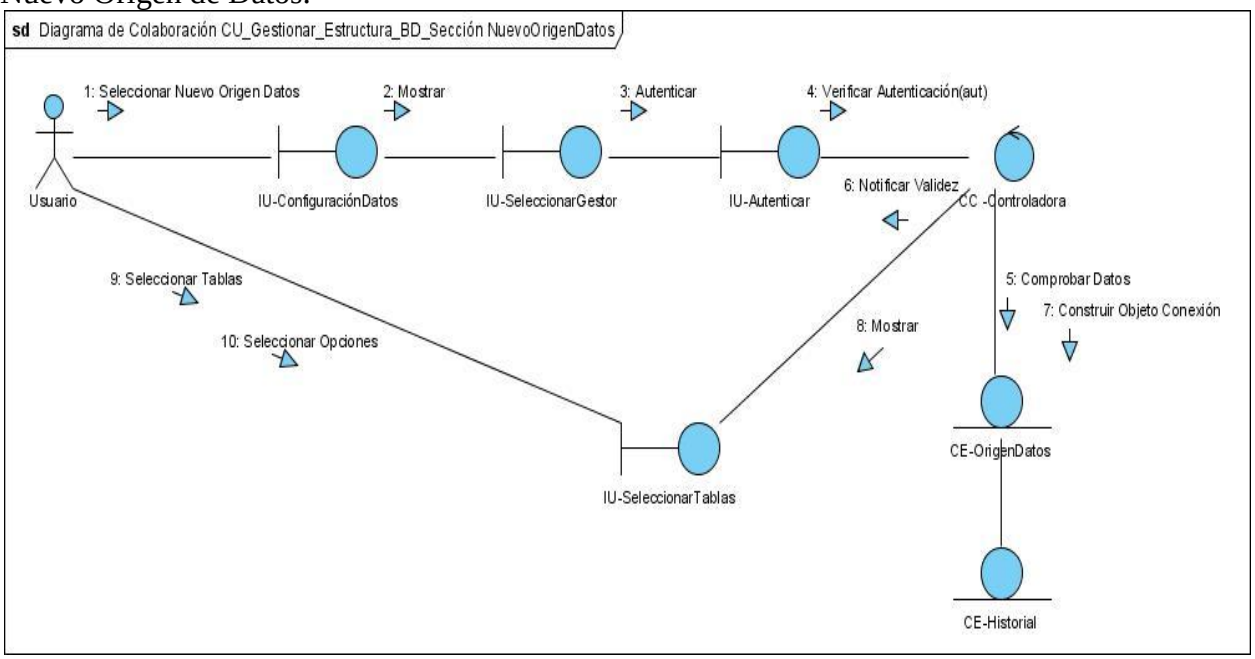


Figura 13 Diagrama de Colaboración CU Conexión Actual

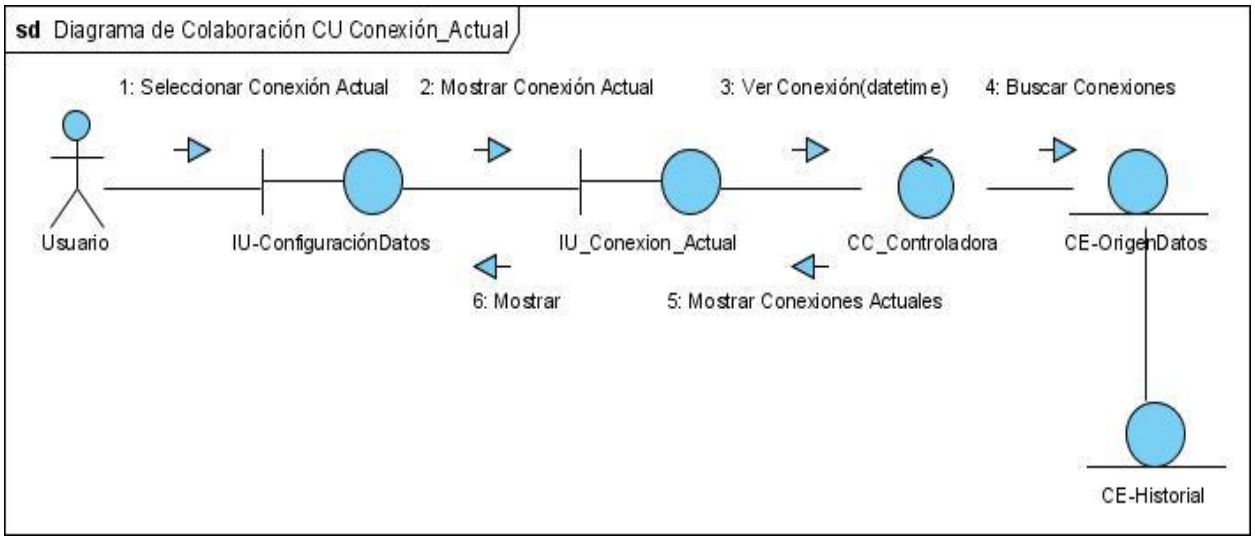



Figura 14 Diagrama de Colaboración CU Mostrar Historial

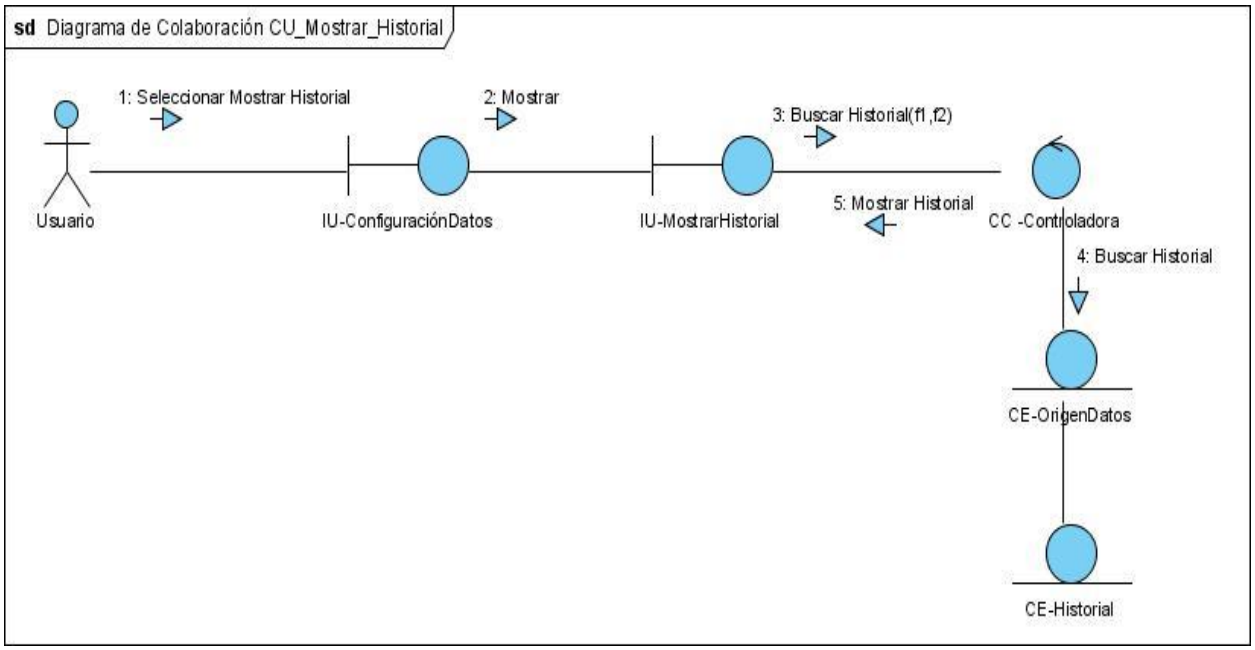


Figura 15 Diagrama de Colaboración CU Autenticarse para Access

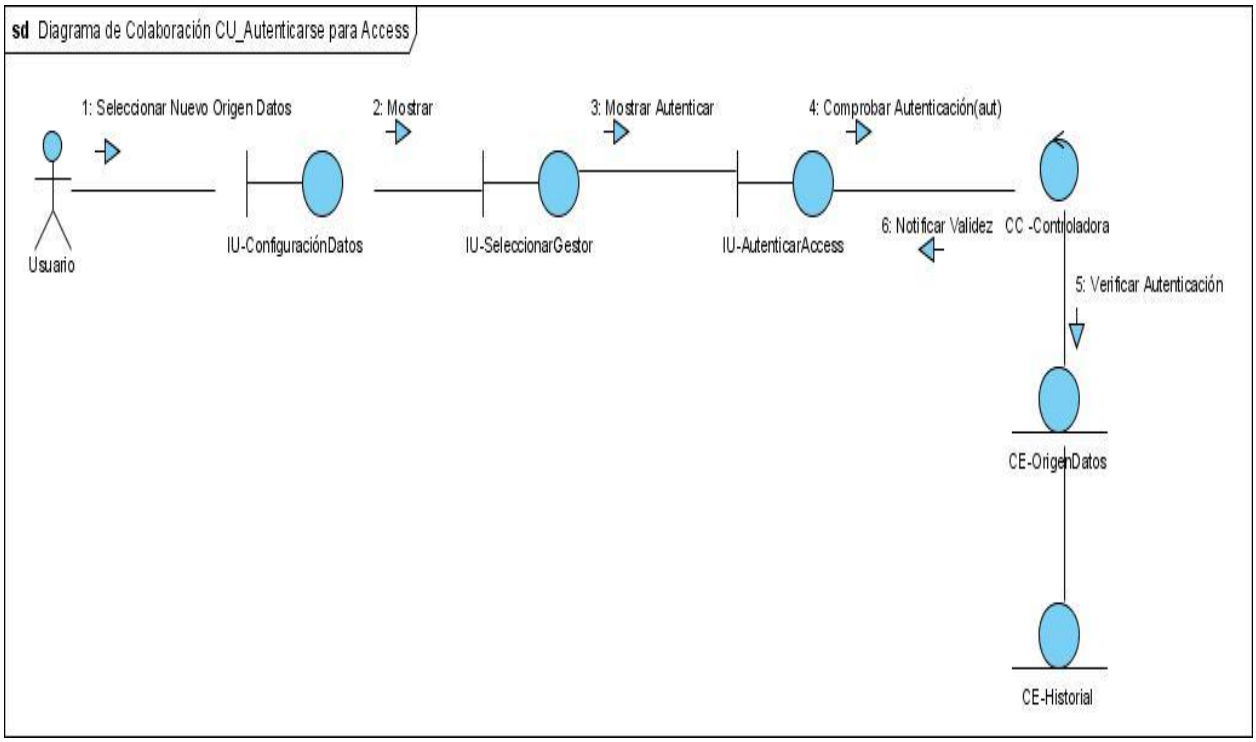



Figura 16 Diagrama de Colaboración CU Autenticarse para MySQL

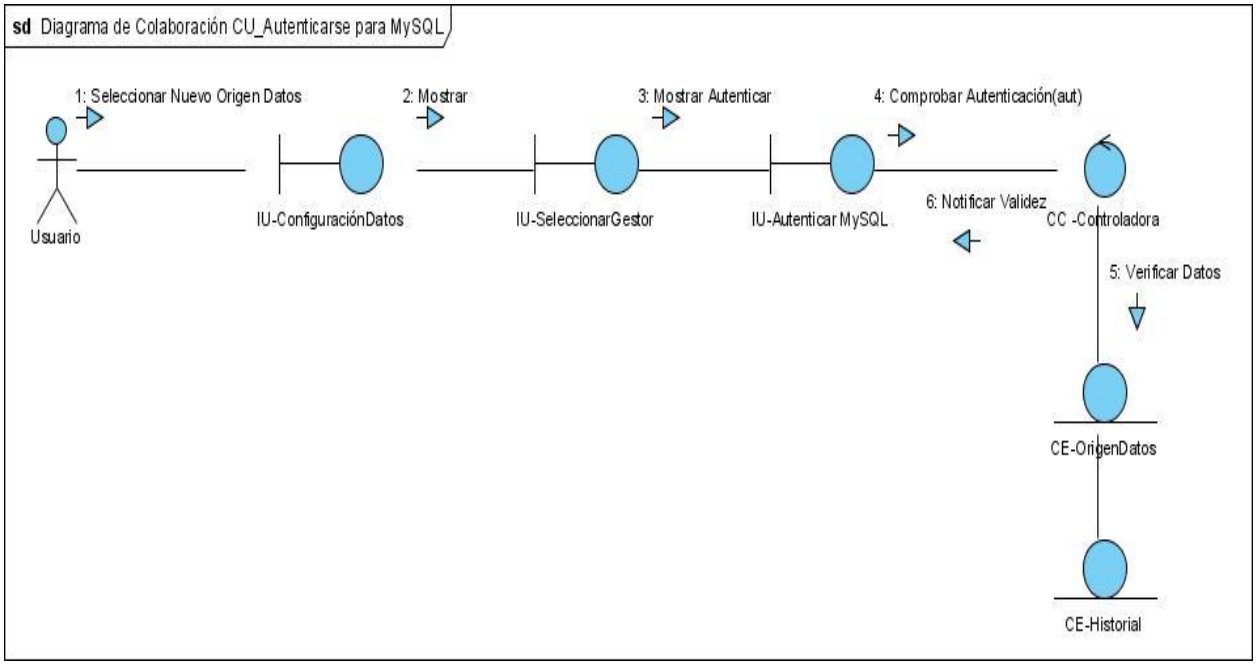


Figura 17 Diagrama de Colaboración CU Autenticarse para Oracle

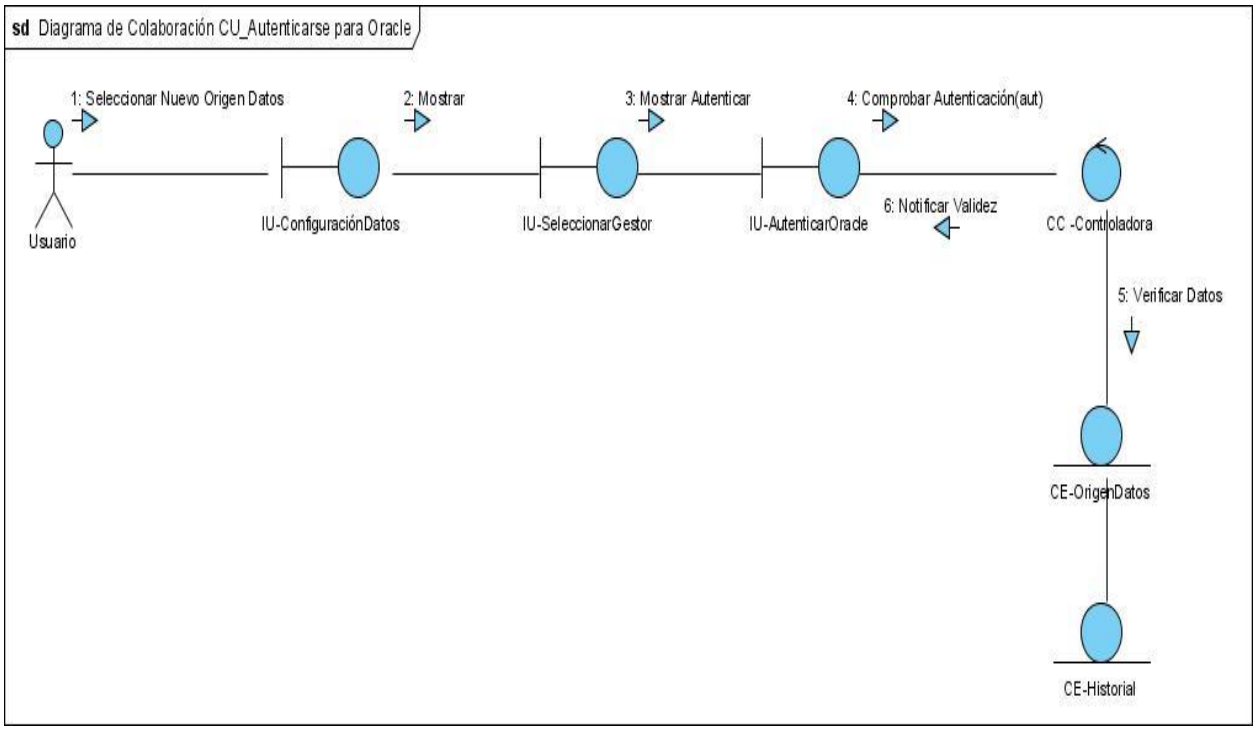



Figura 18 Diagrama de Colaboración CU Autenticarse para Postgres

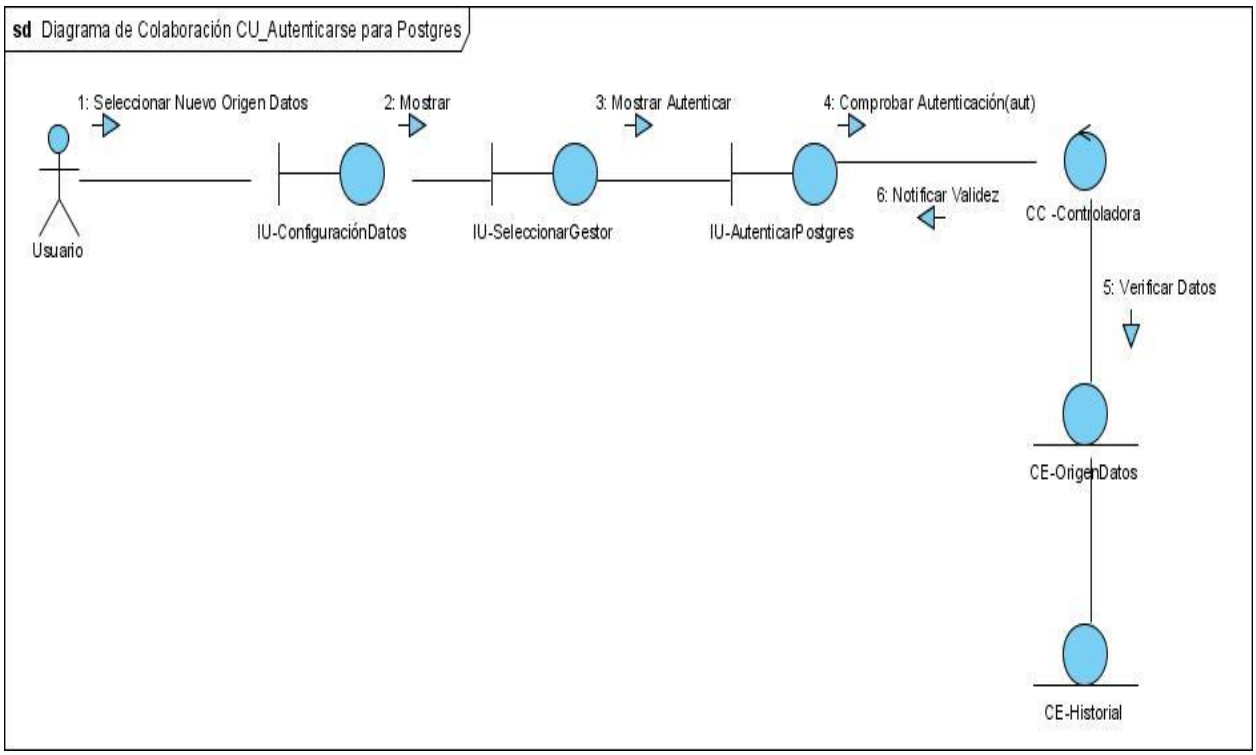


Figura 19 Diagrama de Colaboración CU Autenticarse para SQL Server

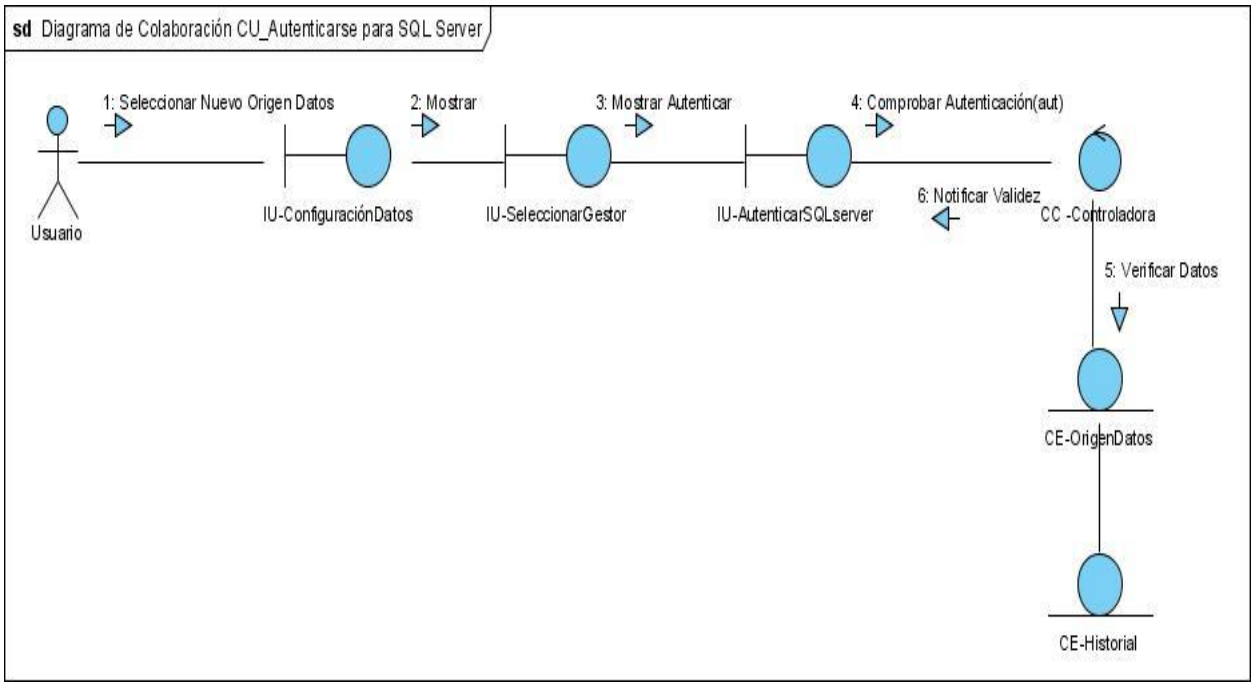


Figura 20 Diagrama de Secuencia CU Gestionar Estructura de la Base de Datos, Sección Nuevo Origen de Datos.

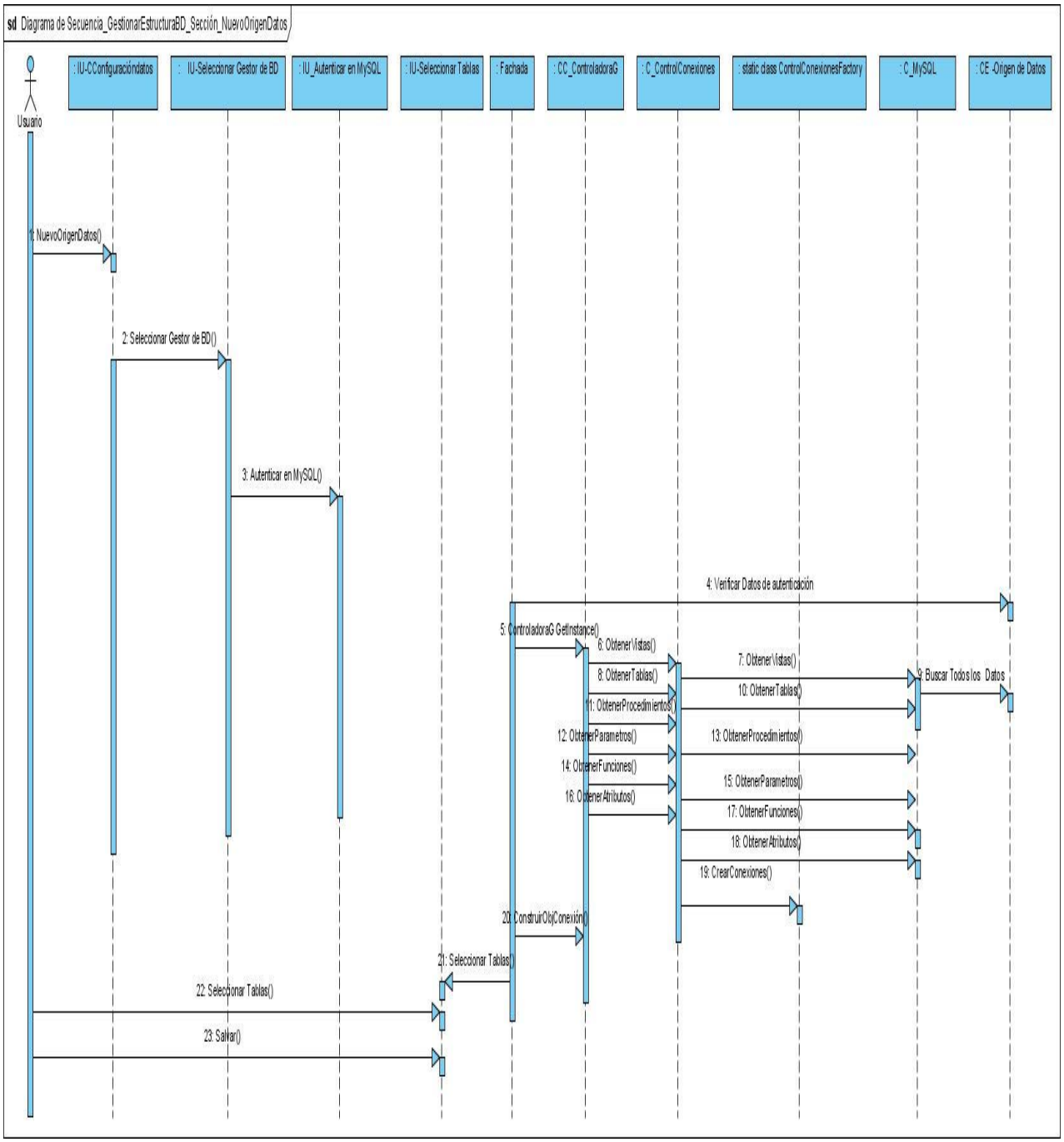


Figura 21 Diagrama de Secuencia CU Gestionar Estructura de la Base de Datos, Sección Gestionar Origen de Datos.

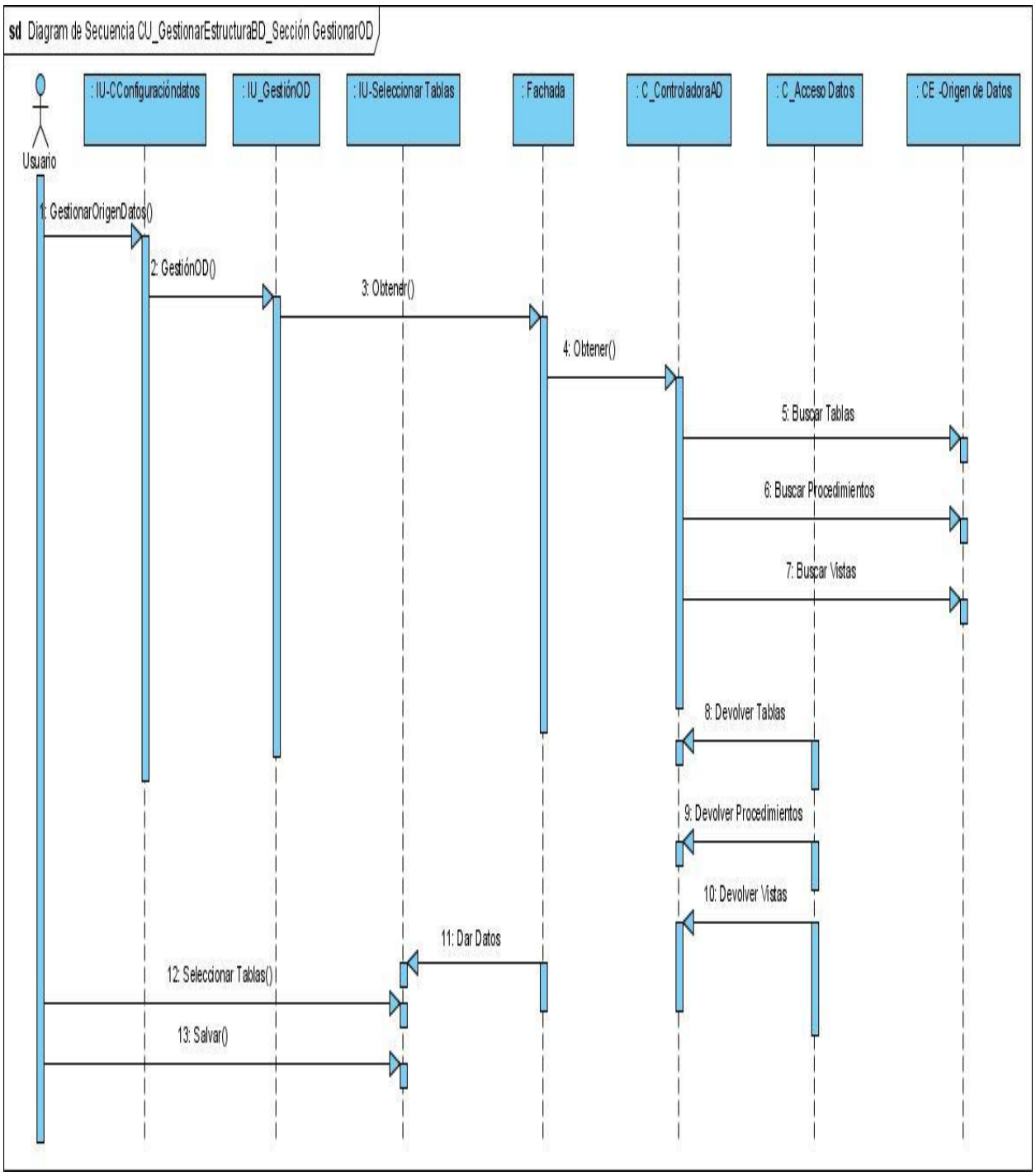


Figura 22 Diagrama de Secuencia CU Mostrar Historial

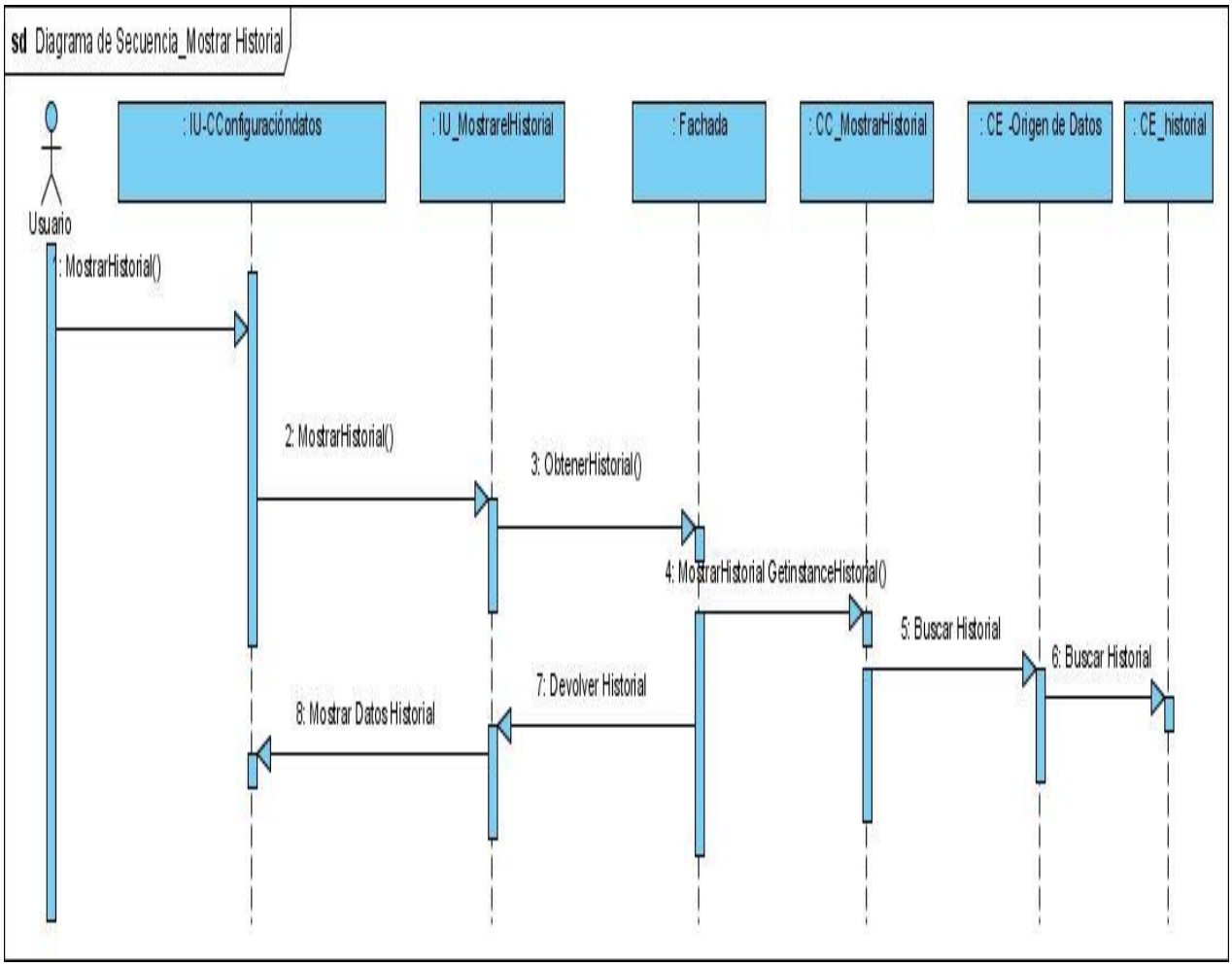


Figura 23 Diagrama de Secuencia CU Conexión Actual

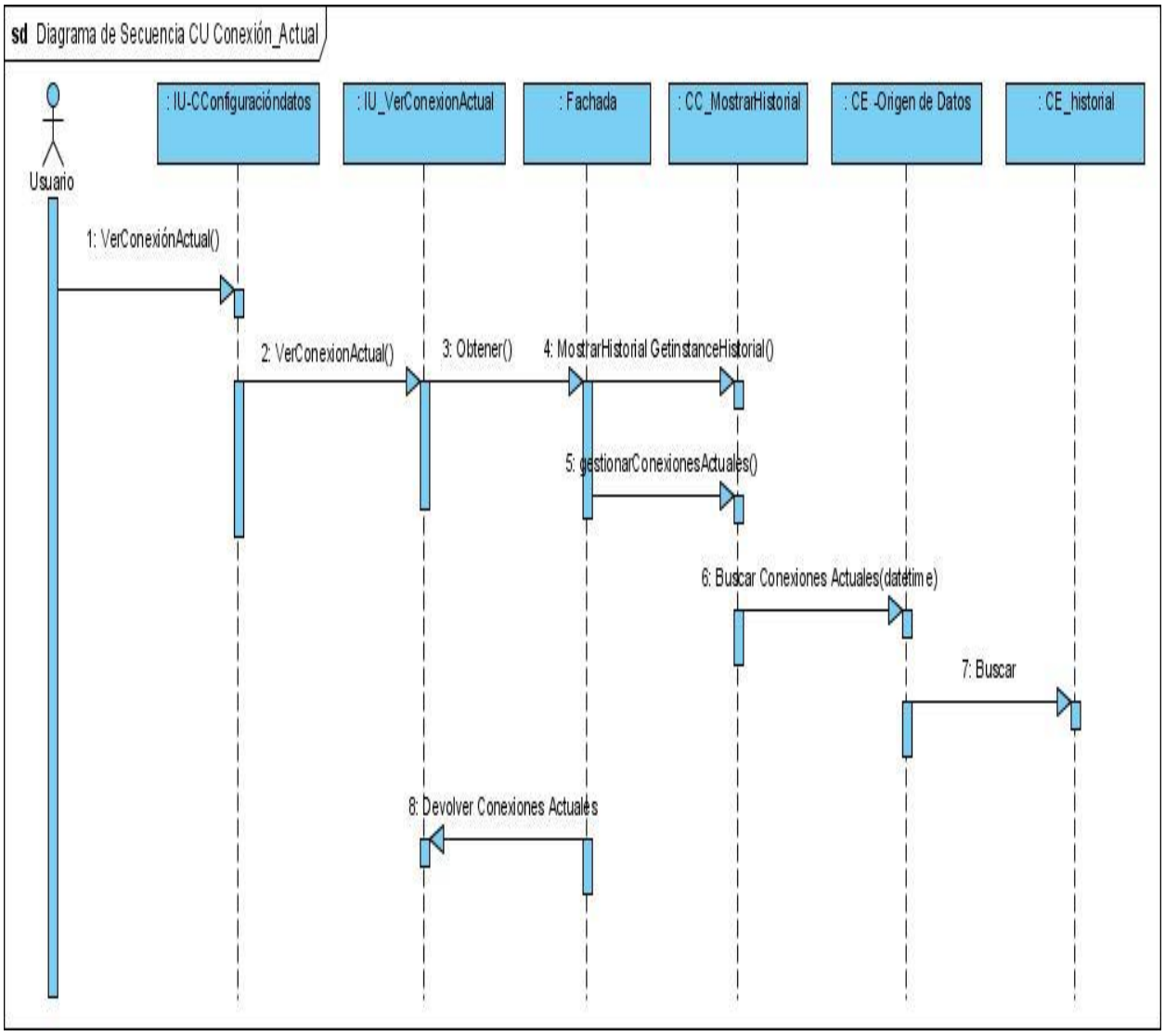


Figura 24 Diagrama de Clases del Análisis CU Conexión Actual

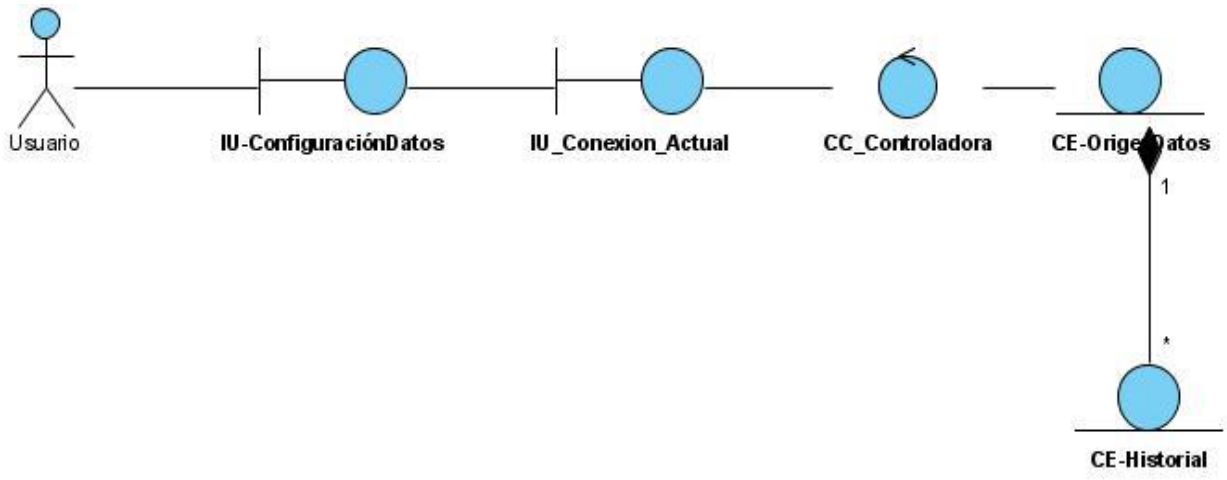



Figura 25 Diagrama de Clases del Análisis CU Autenticarse para Access

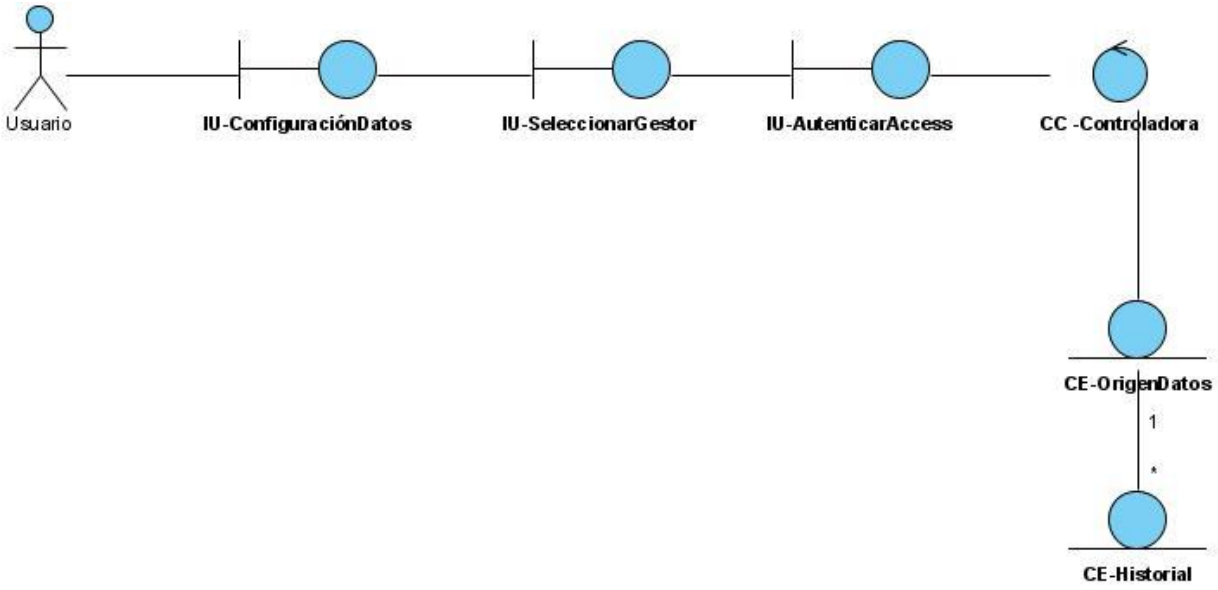


Figura 26 Diagrama de Clases del Análisis CU Autenticarse para MySQL

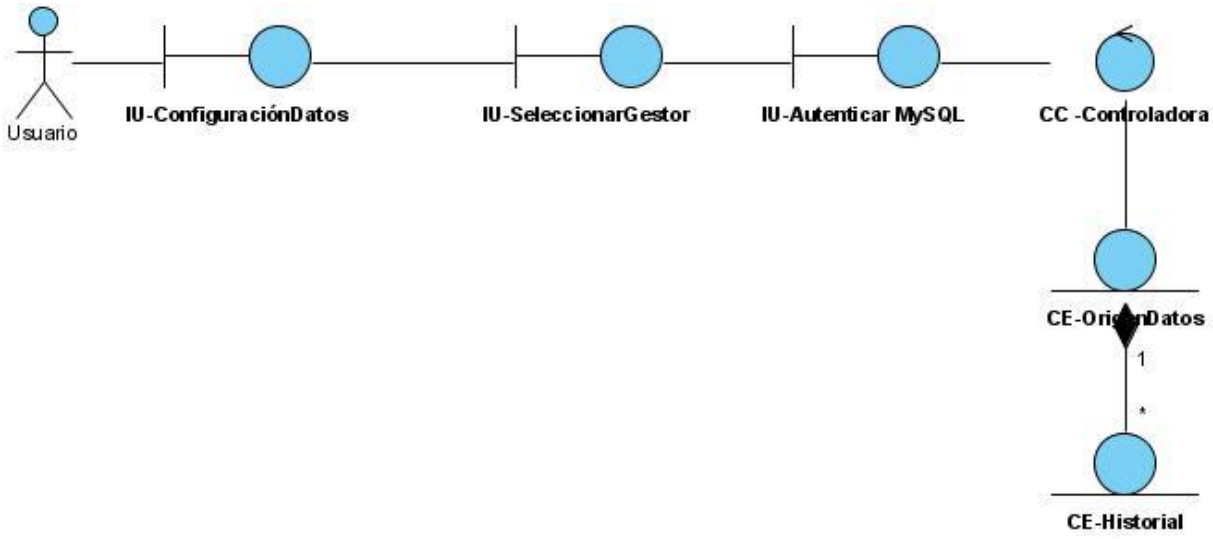



Figura 27 Diagrama de Clases del Análisis CU Autenticarse para Oracle

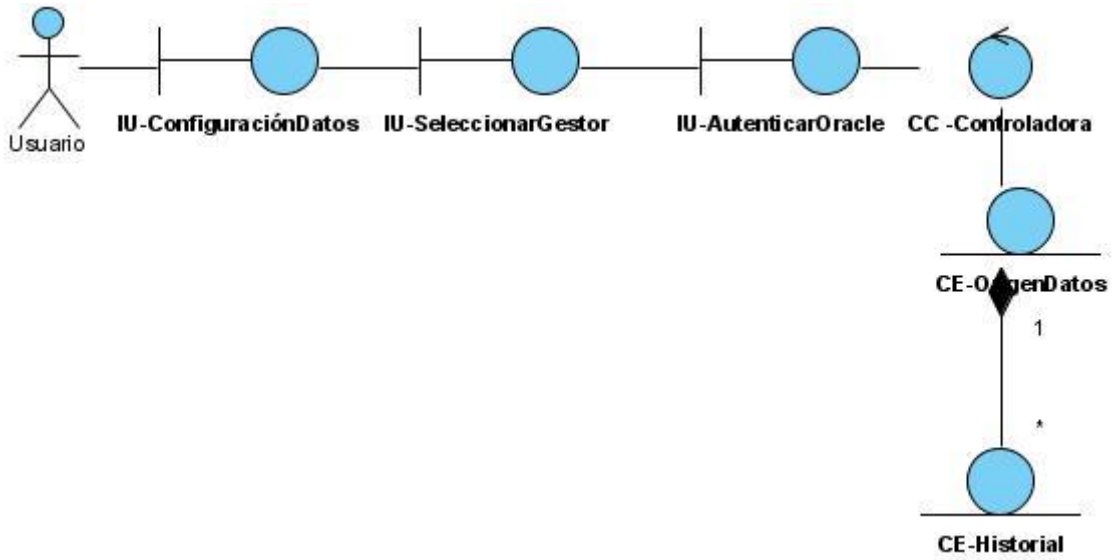


Figura 28 Diagrama de Clases del Análisis CU Autenticarse para Postgres

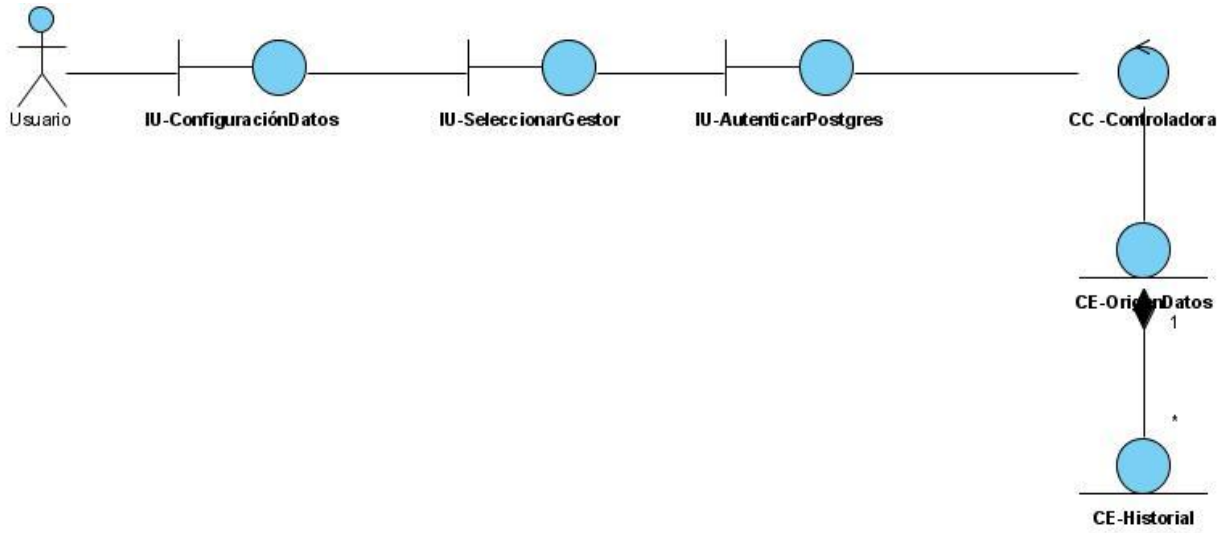



Figura 29 Diagrama de Clases del Análisis CU Autenticarse para SQL Server

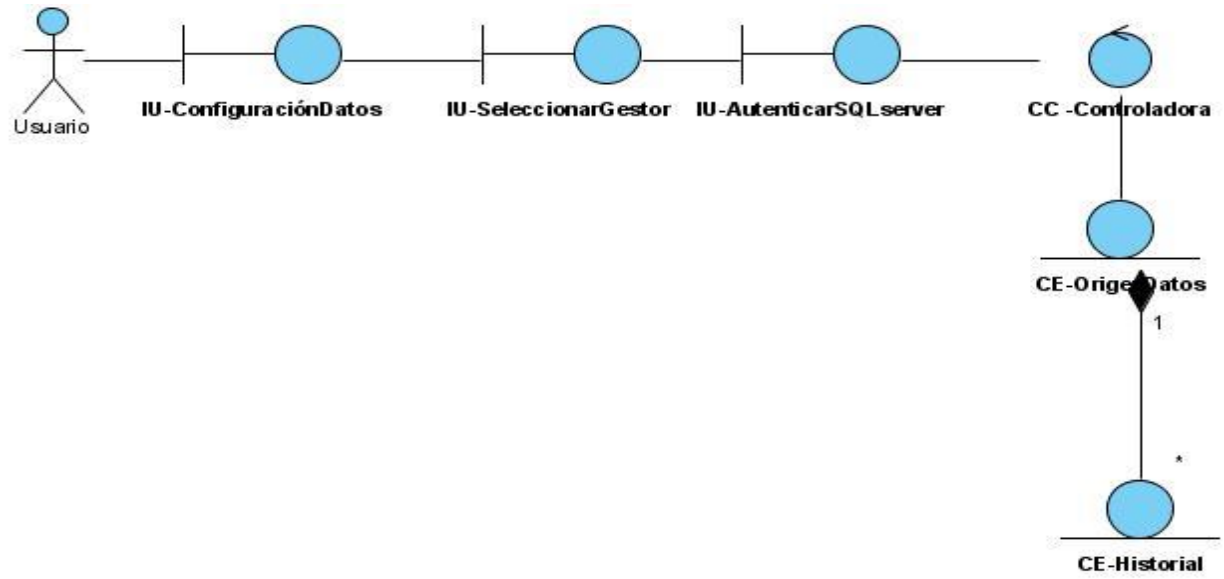


Figura 30 Diagrama de Clases del Diseño del CU Mostrar Historial

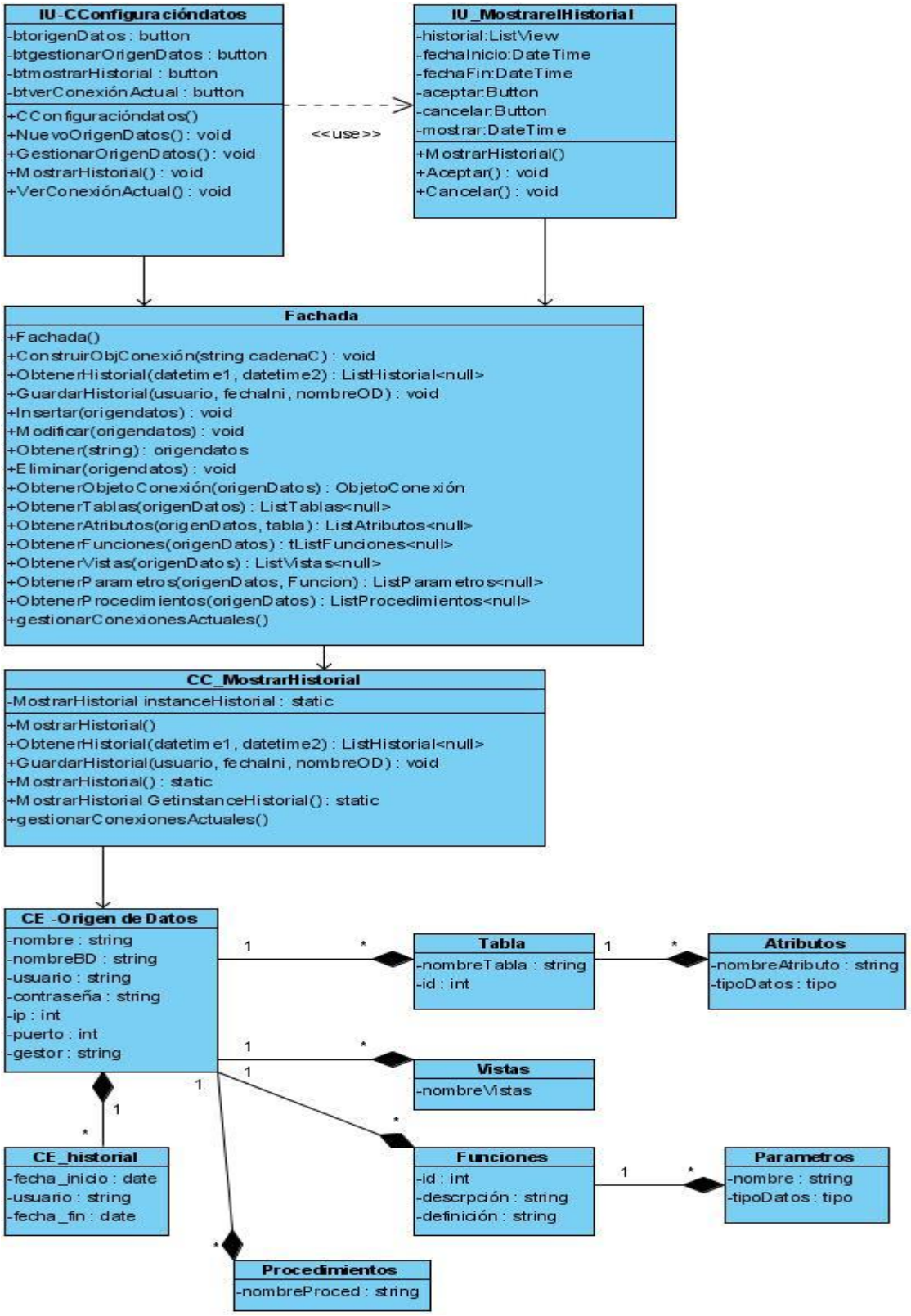


Figura 31 Diagrama de Clases del Diseño del CU Conexión Actual

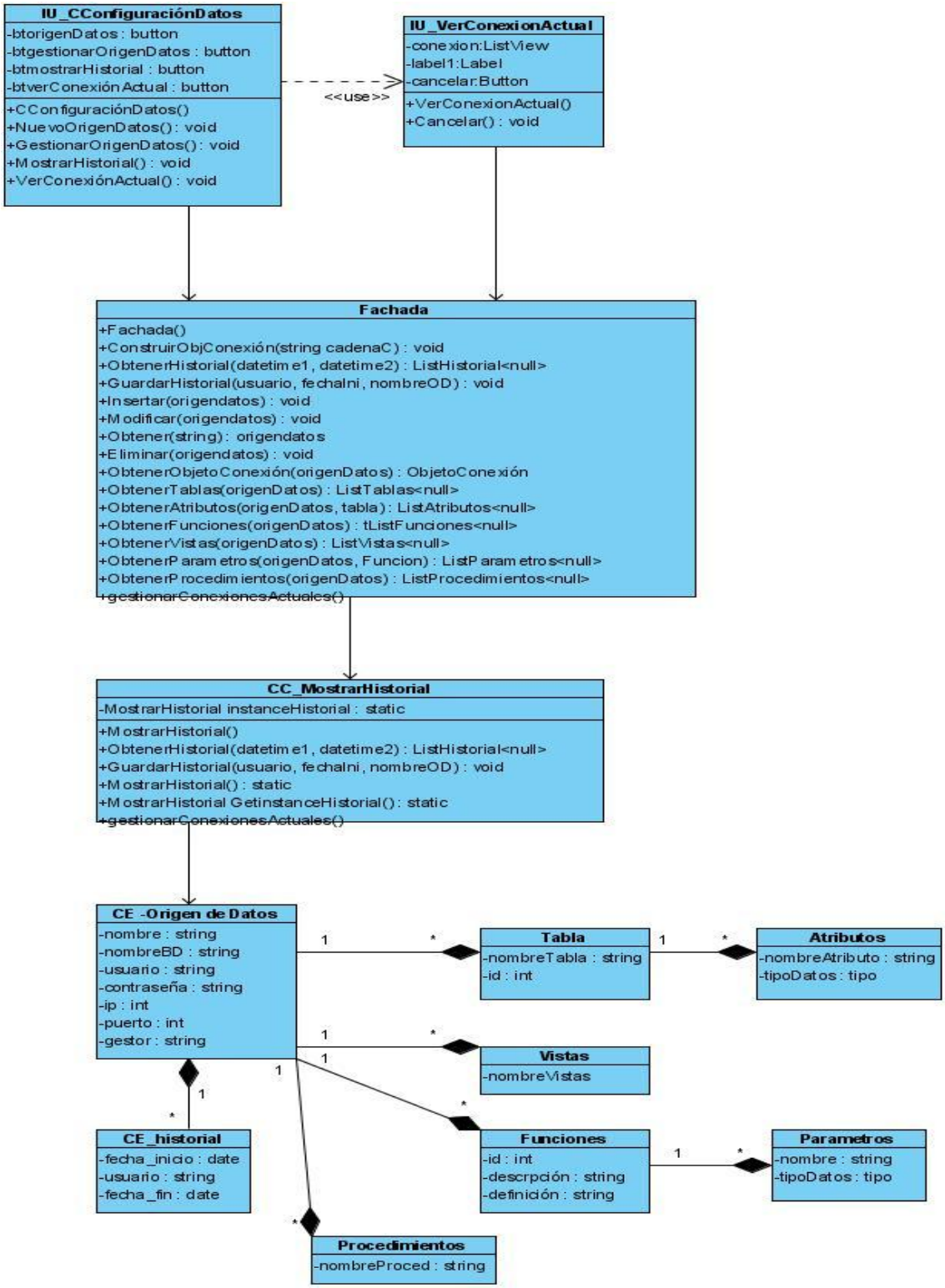
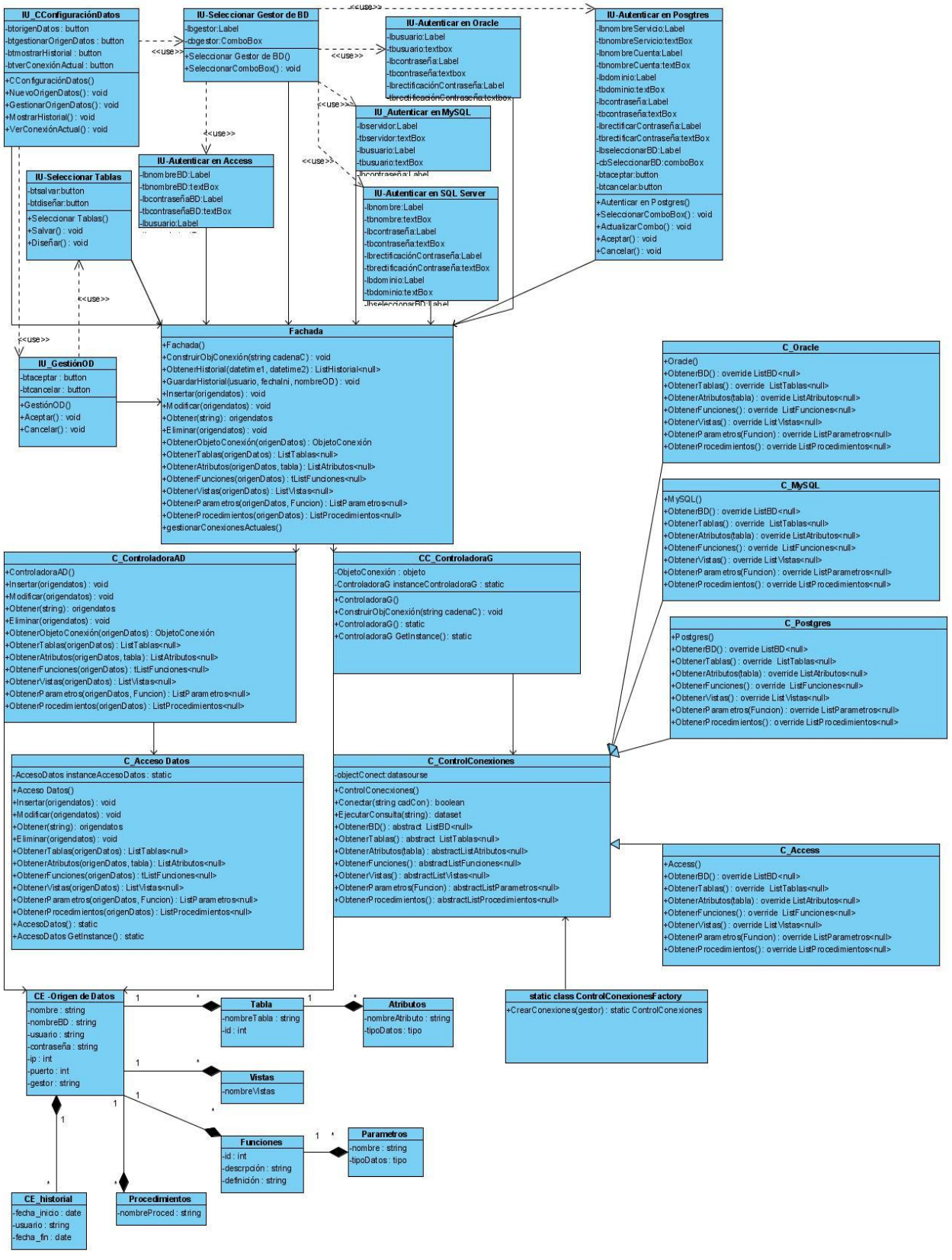


Figura 32 Diagrama de Clases del CU Gestionar Estructura de la Base de Datos



# Tablas

**Tabla 1.** Metodologías Ágiles vs Metodologías Tradicionales

| Metodologías ágiles | Metodologías tradicionales |
|---|---|
| Basadas en heurísticas provenientes de prácticas de producción de código | Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo. |
| Especialmente preparados para cambios durante el Proyecto. | Cierta resistencia a los cambios |
| Impuestas internamente (por el equipo) | Impuestas externamente |
| Proceso menos controlado, con pocos principios. | Proceso mucho más controlado, con numerosas políticas/normas |
| No existe contrato tradicional o al menos es bastante flexible. | Existe un contrato prefijado |
| El cliente es parte del equipo de desarrollo | El cliente interactúa con el equipo de desarrollo mediante reuniones |
| Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio | Grupos grandes y posiblemente distribuidos. |
| Pocos artefactos | Más artefactos |
| Pocos roles | Más roles |
| Menos énfasis en la arquitectura del software | La arquitectura del software es esencial y se expresa mediante modelos |