Universidad de las Ciencias Informáticas

Facultad 3

Título: Análisis y Diseño de un Sistema para la Generación de Reportes

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

Autores: Llilian Durand Martínez

Osmany Becerra González

Tutor: Ing. Carlos Yasmany Hidalgo García

Junio 2008

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

Quiero agradecer en primer lugar a mi mamá, por todas las cosas que me ha enseñado desde pequeña, porque fue ella mi primera escuela. Por ser siempre tan preocupada por mis estudios y enseñarme a ser sacrificada y persistente. Por quererme tanto, por sus consejos, su apoyo y sus sacrificios, por darme siempre lo mejor.

A mi tío, por ser más que un tío…mi papá. No te preocupes que siempre voy a estar a tu lado, hoy eres tú el que me ayuda pero cuando tú lo necesites haré lo mismo por ti. Te quiero mucho y te agradezco mucho que seas tan cariñoso y preocupado, y que hayas cargado conmigo desde chiquitica.

A mi papi Macho, el abuelo más lindo del mundo….ojalá la vida te regalara la eternidad para que pudieras estar por siempre a mi lado. Gracias por ser tan bueno conmigo y quererme tanto, siempre te voy a llevar en mi corazón.

A mi padrastro Anthony, gracias por querernos tanto a mi mamá y a mí, por tu alma noble…

A Juani, ya eres de la familia, gracias por quererme como una sobrina y por tu apoyo y ayuda.

A mi familia que es bien cortica, pero que me quiere mucho y sé que están todos bien orgullosos de mí: a mi tía Carmen, mi abuela Hilda, a Prima, a Ari y Adri, Mario, Juanca, Piloto, a mi tío Alberto. Los quiero mucho. A mis vecinas Lucre y Esther, que son como familia.

A Elier, que ha estado a mi lado en este momento tan difícil, y en otros más. Que ha sabido soportar mis malcriadeces, que me ha dado mucho amor y apoyo, y que siempre me dice: “Llili tu lo vas a lograr “. Te quiero tati. A tu familia, me encantó conocerlos.

A mi amiga de siempre Claudia, gracias por estar a mi lado. A su mamá Zoila, que es como mi segunda mamá.

A mis amigas de la Universidad: Mariam, Olgui, Ani y Gini…qué liga hemos hecho!!! Gracias por compartir mis risas y mis lágrimas. A Rauli mi amigo, por ser tan escandaloso y tan servicial. A mi amiga desde pequeña Nely. Espero que seas una gran abogada. A mi grupo desde primer año, por ser tan unido y divertido. Gracias por las fiestas y los estudios.

A Yoandris, por ayudarme con la tesis y ser mi amigo. Gracias por estar siempre que te necesito.

A Osmany, gracias por ser mi compañero de tesis

A Olga y Héctor, gracias por ser tan alegres, y querernos a todas las compañeras de Olgui como a hijas.

A la Revolución, y a la Universidad de las Ciencias Informáticas por ser mi casa estos cinco años.

Llilian

AGRADECIMIENTOS

A mis padres por inculcarme el empeño por ser alguien en la vida.

A mis amistades y amigos que estuvieron ahí y de los que me serví de una manera u otra durante el desarrollo de este trabajo: Ariel, Lizandra, Olguita y Yeni.

A todos aquellos que han contribuido de manera directa al éxito de esta tesis, en especial a Jorge Ernesto y Yoandris.

A Llilian: literalmente, la mitad de este trabajo te lo debo a ti.

A la Revolución y a Fidel.

A los profesores que me formaron.

Y a los que faltan y deberían estar, mas ahora escapan a mi memoria.

A todos, gracias.

Osmany

DEDICATORIA

A la memoria de mi abuela Paula

A mi mamá

A mi abuelo

A tío Jose

A Elier, espero ansiosa que te llegue este gran momento.

Llilian

DEDICATORIA

A mi abuela Mima, que en paz descanse.

A mi madre que nunca me ha faltado.

A mis hermanos Rainer y Reinier: que logren en la vida todo cuanto se propongan.

A mis abuelos, tíos y mis incontables primos.

A mis hermanos sin sangre: el negro Pons, Rafael, Jesús, Yara, Liermes, Enmanuel y Yislenys.

A mami Dania.

Y muy en especial, a mi padre: mi graduación, es mitad tuya.

Osmany

RESUMEN

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

En el presente trabajo se realiza el análisis de un sistema para la generación de reportes, el cual presenta entre sus características la independencia total de la plataforma de desarrollo. La misma posibilita que este sistema sea usado por otros sistemas, sin importar el lenguaje de programación en el cual estos últimos hayan sido desarrollados. Así mismo se realiza el diseño de uno de sus subsistemas principales, el Diseñador de Informes, 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 CONTENIDOS

INTRODUCCIÓN

La difusión masiva de las tecnologías de la información y las comunicaciones en los últimos años del siglo XX, ha generado la llamada revolución informática, que ha dado origen a una nueva época, que se conoce como sociedad de la información. El motor que impulsa la economía pasa de ser los combustibles y la electricidad a la información. La información es hoy el recurso clave de la economía, de las organizaciones, del mundo cultural y de la política.

Yonehi Mashuda[1] en 1980 dio muestras de una excelente capacidad de predicción del futuro, al aseverar que la computadora se aplicaría en una gama muy amplia de necesidades sociales. Todas las actividades humanas involucran de alguna manera el uso de información, es por ello que en muchos países el uso de las tecnologías de la información se encuentra distribuido en muchos sectores de la sociedad (Joyanes, 1997), mientras que otros no tan desarrollados, como nuestra pequeña isla hacen grandes esfuerzos por informatizar la sociedad. Al respecto nuestro Comandante en Jefe en el discurso pronunciado el 7 de marzo del 2006 en conmemoración al aniversario XV de los Joven Club de Computación y Electrónica expresó: “Cuba está entrando en una nueva era en el mundo de la informática, con un sentido universal, como ya lo ha hecho en el campo de la medicina.”

Actualmente la inmensa mayoría de las grandes empresas, por no decir todas, controlan sus actividades fundamentales por medio de sistemas informáticos de diversos grados de complejidad. Los sistemas informáticos, son capaces de controlar y gestionar la información de toda clase de procesos económicos y financieros, procesos de producción, proyectos inversionistas y cualquier otra actividad que el cliente desee informatizar.

En todos los casos, un requisito indispensable de los sistemas informáticos, es brindar la posibilidad de mostrar los resultados de la gestión de la información con diferentes grados de complejidad, exactitud y detalle, según sea requerido por los clientes y, con un grado de personalización de las vistas, tal que satisfaga sus exigencias. Con este fin, existen en el mundo varias herramientas, tanto libres como propietarias, que permiten obtener y analizar información de repositorios de datos heterogéneos y bases de datos; precisamente son estas las herramientas para la generación de reportes. Las mismas

[1] Yonehi Mashuda, The Information Society as a Post-Industrial Society, Tokyo, Institute Inforrnation Society, 1980. Versión en español, La sociedad informatizada como sociedad post-industrial, Madrid, Fundesco, 1984.

posibilitan la materialización de la gestión de la información de una manera relativamente sencilla. Entre estas herramientas se pueden citar el Active Reports, Jasper Reports y Crystal Reports, entre otros.

Nuestra universidad es un centro docente donde la producción ocupa un importante papel, reflejo de ello es la amplia gama de proyectos productivos que en ella se desarrollan, tanto para clientes nacionales como extranjeros. Los sistemas que en estos se producen, en su mayoría brindan una abundante salida de información a modo de reportes, por lo que la utilización de un sistema que brinde soporte a la generación de las mismas es de gran importancia.

Sin embargo, las herramientas empleadas en algunos proyectos con salida de información a modo de reportes, presentan inconvenientes y limitantes, mientras que otros, no se auxilian de ningún sistema que viabilice este proceso, por lo que este resulta lento y agobiante para el desarrollador. Es por ello que resultaría conveniente contar con una herramienta que estandarice y viabilice el proceso de generación de reportes en la Universidad.

Para lograr la construcción exitosa de este sistema es necesario que los desarrolladores tengan una entrada correcta y entendible que les permita implementar el sistema con calidad. Obtener esta entrada implica realizar correctamente las actividades correspondientes a los flujos de trabajo anteriores al de Implementación.

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

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

De esta forma, se define como **objetivo general** del presente trabajo de diploma, realizar el análisis y diseño de un sistema para la generación de reportes, que permita a los desarrolladores implementarlo correctamente, materializando la estandarización de este proceso en la Universidad; logrando satisfacer de esta forma las necesidades de los proyectos productivos.

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

- Obtener el modelo de dominio.
- Especificar los requisitos funcionales y no funcionales del sistema.
- Estructurar el modelo del sistema en función de los requisitos obtenidos.
- Elaborar los artefactos correspondientes al análisis del sistema generador de reportes y el diseño del módulo Diseñador de Informes.

Se define como **objeto de estudio** el Proceso de Desarrollo de Software, centrando la investigación en el Análisis y Diseño de un sistema para la generación de reportes, identificando el mismo como **campo de acción.**

La **hipótesis** que se plantea es que si se analiza y diseña correctamente un sistema para la generación de reportes se podrán obtener los artefactos de entrada correctos y entendibles que permitirán a los desarrolladores su implementación, facilitando así la estandarización del proceso de generación de reportes en la Universidad y la satisfacción de las necesidades de los proyectos productivos.

Con vista a dar cumplimiento a los objetivos planteados, se definen las siguientes **tareas de investigación**:

- Revisión de la bibliografía existente acerca del tema de análisis y diseño para lograr un desarrollo exitoso.
- Estudio del estado actual de los sistemas generadores de reportes.
- Selección de la metodología y herramientas adecuadas para el desarrollo.
- Realización de entrevistas a personas vinculadas con la generación de reportes en los diferentes proyectos productivos, con el objetivo de identificar las características y funcionalidades principales que debe ofrecer la herramienta.
- Estudio de los patrones de diseño que puedan ser empleados en la solución de nuestro problema.

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

Métodos teóricos:

- **Analítico – Sintético:** Para entender fenómenos relacionados con el proceso de generación de reportes, así como las tendencias actuales del desarrollo de software, a partir de la documentación analizada, lo que permitió la extracción de elementos importantes relacionados con el objeto de estudio.
- **Histórico – Lógico:** Permitió constatar teóricamente la evolución del proceso de generación de reportes, así como las herramientas actuales que permiten la materialización de este proceso.

Métodos empíricos:

- **Entrevista:** Para obtener información de las características deseables para un sistema generador de reportes, así como algunos detalles sobre el proceso de generación de reportes en los proyectos productivos de la Universidad.

Aportes prácticos esperados del trabajo:

Se espera conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero. Obtener un diseño flexible y entendible que sirva de abstracción a la implementación, y que pueda ser utilizado como una entrada fundamental a este flujo de trabajo.

Resultados esperados:

- Especificación de los requisitos del software
- Especificación de los CUS[2]
- Modelo de análisis
  - ✓ Diagrama de clases del análisis
  - ✓ Diagramas de colaboración del análisis
- Modelo de diseño del subsistema Diseñador de Informes
  - ✓ Diagrama de clases del diseño del subsistema Diseñador de Informes
  - ✓ Diagramas de secuencia del diseño del subsistema Diseñador de Informes

[2] Casos de uso del sistema

Estructuración del contenido con una breve descripción de sus partes

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

Capítulo 1: En este capítulo se realiza un estudio del Proceso de desarrollo de software, por lo que se hace un análisis de las diferentes metodologías y tecnologías que se pueden emplear en la solución, seleccionando la más apropiada para llevar a cabo dicho proceso con éxito y calidad. Se realiza además un estudio del estado del arte de las principales herramientas que existen en el mundo para la generación de reportes tanto libres como propietarias. Se define además la labor del analista de sistemas y el diseñador. Se realiza un estudio del estado del arte de los patrones de casos de uso, patrones de diseño y patrones GRASP que pueden ser empleados en la solución.

Capítulo 2: En este capítulo como parte de la propuesta de solución, se describen las características del sistema, se presenta el modelo de dominio, así como el glosario, elaborado con los principales conceptos del dominio del problema y las relaciones que entre ellos se establecen. Se realiza la especificación de los requisitos funcionales y no funcionales del sistema, determinándose a su vez los casos de uso y los actores, para conformar el DCUS[3]; teniendo en cuenta ciertos patrones de casos de uso. Para cada uno de los casos de uso se elabora una descripción en formato expandido.

Capítulo 3: Este capítulo contiene lo referente al análisis y diseño del sistema. Como parte de la solución se desarrollan los diagramas de clases del análisis para cada uno de los casos de uso, los diagramas de clases del diseño del subsistema Diseñador de Informes, así como los diagramas de interacción correspondientes al análisis y al diseño. En el mismo se exponen los patrones de diseño empleados en la solución, con una breve descripción de sus características y el propósito de su uso en el sistema.

Capítulo 4: 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.

[3] Diagrama de Casos de Uso del Sistema

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se hace referencia al proceso de desarrollo de software, así como las metodologías de desarrollo de software, tecnologías y tendencias actuales que pueden ser útiles en el desarrollo de la propuesta de solución. En dependencia de la metodología seleccionada se realiza un estudio del estado del arte del análisis y diseño, y los principales artefactos que se obtienen como resultado de este, así como los roles que en este intervienen. Se realiza un estudio del estado del arte de las principales herramientas para la generación de reportes que existen en el mundo, de los patrones de casos de uso y de los patrones de diseño.

1.2. Ingeniería del 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. 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.

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.”

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)

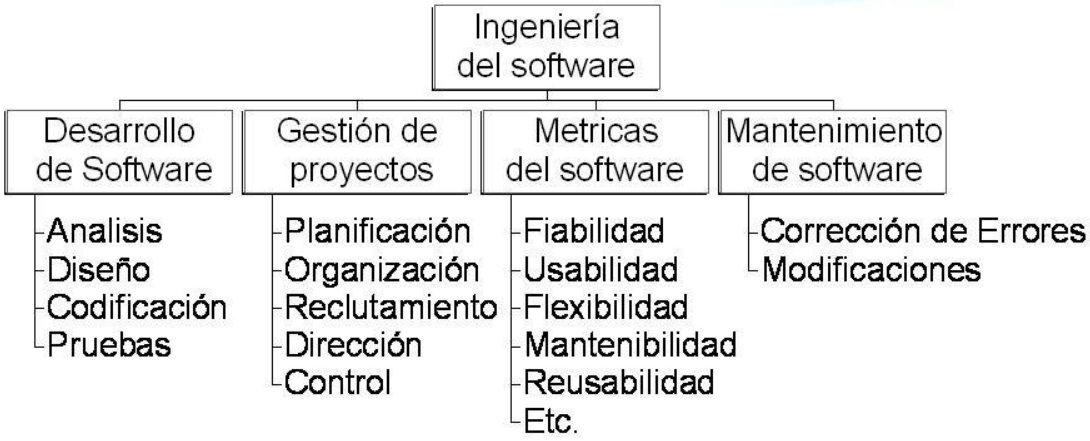


Figura 1.1. Modelo de la Ingeniería de Software (Thayer, 1988) (Montesa Andrés)

Después de estudiar varias definiciones de Ingeniería de Software podemos concluir que las buenas prácticas de la Ingeniería de Software permiten crear equipos de alto rendimiento que producen proyectos más exitosos porque están en plazo, en presupuesto 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 de software

Un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. (Rumbaugh, y otros, 2004)

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, 2000)

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 recogen datos sobre el proceso de desarrollo del software.
- 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.

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, y otros, 2007)

Existen en la actualidad múltiples metodologías que 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, y otros)

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

1.3.2. RUP

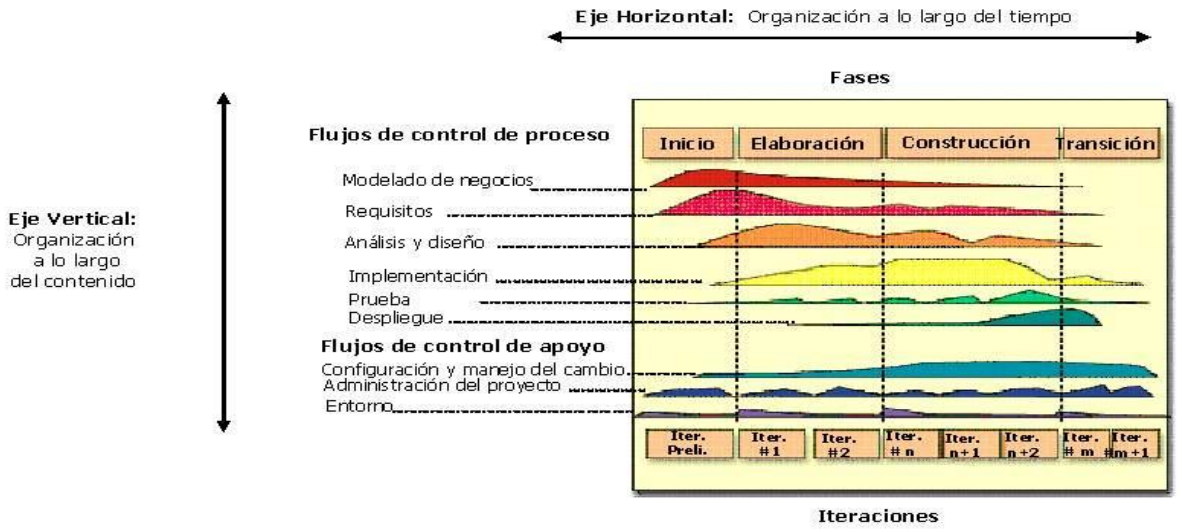


Figura 1.2. Flujos de trabajo y fases de la metodología RUP

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. (Rumbaugh, y otros, 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 beta, o sea, un número reducido de usuarios con experiencia prueba el producto e informa de defectos y deficiencias. Esta fase conlleva a actividades como la fabricación, formación del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección de defectos que se encuentren tras la entrega. (Rumbaugh, y otros, 2004 pág. 11 y 12)

Como ilustra la figura 1.2 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.

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. (Rumbaugh, y otros, 2004 pág. 12)

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. (Rumbaugh, y otros, 2004 pág. 5)

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. (Rumbaugh, y otros, 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)

- **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, y otros, 1992)
- **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. (Rumbaugh, y otros, 1999)
- **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.

Fases de la metodología XP:

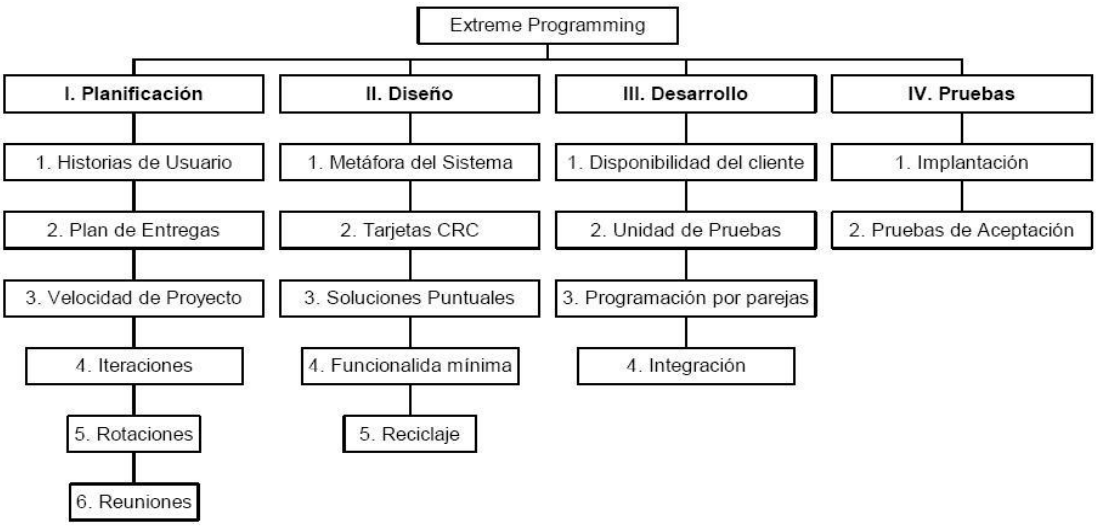


Figura 1.3. Fases de la metodología XP (Escribano, 2002)

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. (Palmero Sánchez, y otros, 2007)

1.3.4. Justificación de la metodología a utilizar.

Después de estudiar 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 podemos 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 sistema para la generación 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 esta es más adecuada.

Con respecto a la metodología RUP el Dr. Xavier Ferré Grau[4] 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, y otros, 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, y otros, 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. (Rumbaugh, y otros, 1998)

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.

[4] Doctor en Informática. Universidad Politécnica de Madrid

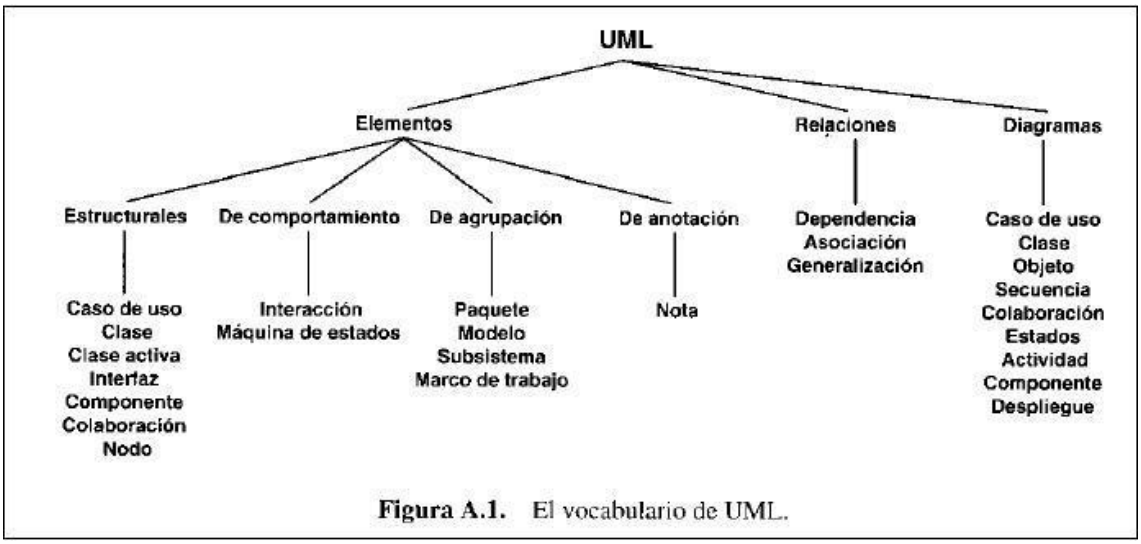


Figura A.1. El vocabulario de UML.

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 actuales y futuras.

- Ampliamente utilizado por la industria desde su adopción por OMG[5].
- 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, Mayo 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. Herramientas 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, y otros, 1998 págs. 93-99)

De acuerdo con Kendall y Kendall (Kendall, y otros, 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.

[5] Object Management Group

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, y otros, 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 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 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.

1.6. Ingeniería de Requisitos

La satisfacción del cliente se considera la mejor métrica de la calidad de un sistema. (Pazos Arias, 2000). Lograr una comunicación efectiva 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.
[...]Ninguna otra parte del trabajo afecta más negativamente al sistema final si se realiza de manera incorrecta. Ninguna otra parte es más difícil de rectificar después […][Brooks 1995] (García Ávila, y otros, 2007)

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)

1.6.1. Técnicas empleadas en la captura de requisitos

La captura de requisitos es la actividad mediante la que el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema. (Díez, 2001). Por la complejidad que todo esto puede implicar, la ingeniería de requisitos define técnicas que permiten hacer este proceso de una forma más eficiente y precisa. Estas técnicas son:

- Entrevistas
- JAD[6]
- Tormenta de ideas
- Casos de Uso
- Cuestionarios

[6] Join Application Development-Desarrollo Conjunto de Aplicaciones

- Comparación de Terminología

Entrevistas

Resultan una técnica muy aceptada dentro de la ingeniería de requisitos y su uso está ampliamente extendido. Las entrevistas le permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada. 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, y otros, 2002)

JAD

Esta técnica resulta 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.

Esta técnica presenta una serie de ventajas frente a las entrevistas tradicionales, ya que ahorra tiempo al evitar que las opiniones de los clientes se tengan que contrastar por separado, pero requiere un grupo de participantes bien integrados y organizados. (José Escalona, y otros, 2002)

Tormenta de ideas

Es también una técnica de reuniones en grupo, cuyo objetivo es que los participantes muestren sus ideas de forma libre. Consiste en la mera 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. Además suele ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del mismo, por lo que suele aplicarse en los primeros encuentros. (José Escalona, y otros, 2002)

Casos de uso

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, y otros, 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, y otros, 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.

1.7. Patrones

El planteamiento de formalizar soluciones a distintas situaciones, de modo que puedan ser entendidas por otros profesionales, es lo a lo que se llama patrones. Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto.

Un patrón es un modelo que podemos 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. (Oktaba). Los patrones permiten y han permitido en diferentes áreas del conocimiento humano rehusar la esencia de la solución de un problema al enfrentar nuevos problemas similares.

1.7.1. 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. Los patrones de casos de uso capturan buenas prácticas en el modelo de casos de uso, son usados: como plantillas para saber como estructurar el modelo de casos de uso, en el caso de los estructurales, o para saber como deben ser organizadas las descripciones de los casos de uso, en el caso de los descriptivos.

Entre los patrones de casos de uso se encuentran los siguientes:

- 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: Extensión

Descripción: consiste en dos casos de uso y una relación de extensión entre ellos. El caso de uso extendido es concreto, esto significa que puede ser instanciado por su cuenta o como una extensión del caso de uso base.

Aplicación: Este patrón es aplicable cuando un flujo puede extender el flujo de otro caso de uso, así como ser realizado por su cuenta.

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)

1.7.2. Patrones de diseño

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. Los patrones de diseño identifican las clases participantes y las instancias, sus papeles y

colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se enfoca sobre un particular diseño orientado a objetos. Se describe cuándo se aplica, las características de otros diseños y las consecuencias y ventajas de su uso. (Martínez Juan)

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular.

Los patrones de diseño se pueden clasificar atendiendo a dos criterios, el primero el propósito, refleja que hace un patrón. El segundo el alcance, especifica si el patrón se aplica especialmente a las clases o a los objetos. Esta clasificación se muestra en la siguiente tabla:

		PROPÓSITO		
		Creacionales	**Estructurales**	**Comportamiento**
ALCANCE	Clase	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

(Gamma, y otros, 1998)

1.7.2.1. Patrones GRASP[7]

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. 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[8] tanto que produzca los resultados negativos propios de un alto acoplamiento. El bajo acoplamiento soporta el diseño de clases más

[7] General Responsibility Assignment Software Patterns(Patrones Generales de Software para Asignar Responsabilidades)

[8] Medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.

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)

1.8. Análisis y diseño en RUP

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. Un flujo de trabajo es una secuencia de actividades que producen un resultado de valor observable. (Rational Software Corporation, 2003)

El análisis y diseño como se mencionó en el epígrafe 1.3.2 es uno de los 9 flujos de trabajo de la metodología RUP y será estudiado en este epígrafe.

[9] Sinónimo de evento del sistema, es un evento de alto nivel generado por un actor externo. Se asocia a las operaciones del sistema

[10] Objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

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.

El propósito de este flujo de trabajo es:

- Transformar los requerimientos en un diseño de lo que será el sistema.
- Desarrollar una Arquitectura sólida para el sistema.
- Adaptar el diseño para que permita un correcto desempeño en el entorno de implementación. (Rational Software Corporation, 2003)

1.9.2. 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. (Rational Software Corporation, 2003)

1.9.2.1. 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. (Rational Software Corporation, 2003)

1.9.2.2. 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. (Rational Software Corporation, 2003)

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.

Santos define las funciones del analista de sistemas para la década de los ochenta como sigue:

"...el analista de problemas en computación deberá conocer procedimientos para indagar sobre lo existente y para saber proponer un verdadero sistema racionalizado, pero también deberá conocer sobre modernos sistemas de información, base del diseño, sobre todo en computación... Estos últimos factores son los que justifican tal especialidad, porque realmente debieron existir los analistas de sistemas, aunque no hubiera computadores, siempre hubo sistemas para organizar, que posiblemente no se difundieron porque no existieron en importancia esos dos factores que hoy prevalecen: el computador y la información." (Santos, 1980)

1.9.2. 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. (Rational Software Corporation, 2003)

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 (Rational Software Corporation, 2003)
- 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.

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. (Rumbaugh, y otros, 2004)

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. (Rational Software Corporation, 2003)

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

Tradicionalmente, la información había sido entregada en reportes impresos en papel. Aún cuando los reportes en papel no pierden total vigencia, cada vez más y más empleados calificados necesitan datos en otros formatos, más fáciles de usar, y que les puedan ofrecer un mayor nivel de detalle y flexibilidad. Los empleados necesitan hacer un análisis más sofisticado y necesitan disponer de la información más rápido, también necesitan medios más flexibles para distribuirla. Los consumidores de información necesitan estar en capacidad de interactuar con los resultados obtenidos, para poder tomar sus propias decisiones, basándose en los datos. Aquí radica precisamente la importancia de los sistemas generadores de reportes.

En aras de conocer las principales características, así como desventajas de algunos sistemas generadores de reportes, se realizó un estudio del estado del arte de los mismos, basándose en varios criterios como: tipo de herramienta, plataforma, soporte para gráficos, imágenes y sub-reportes, orígenes de datos soportados, y otros aspectos relevantes.

Crystal Reports

Tipo	Plataforma	Características	Desventajas
Paquete y/o Sistema	Propietaria	✓ Soporta gráficos, imágenes y sub-reportes ✓ Soporta una amplia variedad de	✓ Es propietario ✓ Para modificar el reporte es preciso

		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 ✓ Viene integrado a algunas herramientas de desarrollo como Visual Studio.NET. ✓ Cuenta con años de experiencia y desarrollo.	recompilar el proyecto donde se encuentra.

Active Reports

Tipo	Plataforma	Características	Desventajas
Sistema	Propietaria	✓ 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 ✓ Cuenta con años de experiencia y desarrollo.	✓ Es propietario

Reporting Service

Tipo	Plataforma	Características	Desventajas
Paquete	Propietaria	✓ 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	✓ Es propietario ✓ Dependiente de la plataforma .NET ✓ Fuente de datos restringida a servidores de Microsoft SQL Server

Jasper Reports

Tipo	Plataforma	Características	Desventajas
Paquete y/o Sistema	Libre	✓ 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 gráficas escrita en Java)	✓ Dependiente de la plataforma Java ✓ Complejo de usar ✓ Deficiente robustez y eficacia para reportes complejos.

Dynamic Jasper

Tipo	Plataforma	Características	Desventajas
Paquete	Libre	✓ Reduce la complejidad del Jasper Report para reportes simples y de mediana complejidad.	✓ Dependiente de la plataforma Java

		✓ 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	✓ Deficiente robustez y eficacia para reportes complejos.

Actuate Enterprise Reporting

Tipo	Plataforma	Características	Desventajas
Sistema	Propietario	✓ Flexible en cuanto al formato y diseño ✓ De fácil manejo y mantenimiento ✓ Potente para reportes internos	✓ Es propietario ✓ Diseñado solo para la Web

Agata Report

Tipo	Plataforma	Características	Desventajas
Paquete y/o sistema	Libre	✓ Multiplataforma (Windows y Linux) ✓ Soporta gráficos, imágenes y sub-reportes ✓ Soporta varios orígenes de datos ✓ Exporta salidas en diferentes formatos (PDF, XML, HTML, etc.) ✓ Es fácil de instalar y ejecutar ✓ Es extremadamente flexible.	✓ Dependiente de la plataforma PHP sobre la que corre directamente y por tanto: ✓ Diseñado solo para la Web usando PHP

Report Manager

Tipo	Plataforma	Características	Desventajas
Paquete y/o sistema	Libre	✓ Multiplataforma (Windows y Linux) ✓ Soporta gráficos, imágenes y sub-reportes	✓ Viene especialmente diseñado para

		✓ Soporta varios orígenes de datos ✓ Soporte para líneas de comandos ✓ Integra un servidor de reportes	usar con Delphi, C++ Builder y Kylix ✓ Su interfaz gráfica es poco intuitiva y poco amigable ✓ Se hace difícil dominarlo y comprender su funcionamiento

Se puede concluir que existen buenas herramientas para la generación de reportes, las cuales soportan gráficos, imágenes y sub-reportes; permiten exportar las salidas en diferentes formatos, como PDF, HTML, XML, etc., y presentan otras características de utilidad.

Sin embargo, todas tienen algunos inconvenientes. Por ejemplo, que para modificar el reporte es necesario recompilar el proyecto donde se encuentra, o que son software propietario, lo que implica que sus licencias son costosas, y esto desde el punto de vista de la situación política y económica de nuestro país es una gran barrera, tal es el caso del Crystal Report y el Active Report. Algunos son dependientes de una plataforma específica, como el Jasper Report y el Agata Report, o dependientes de una única fuente de datos como el Reporting Service, o están diseñados solo para la Web como el Actuate Enterprise Reporting.

Por otro lado, las opiniones que se han encontrado respecto a los sistemas libres para la generación de reportes, como el Jasper Report, Dynamic Jasper y el Agata Report, no son muy buenas, pues aparte de que no están bien documentados, no son, por regla general, lo suficientemente robustos y flexibles para satisfacer los requerimientos de reportes complejos, a diferencia de los bien conocidos Crystal Report y Active Report.

1.11. Conclusiones parciales

La calidad en el desarrollo y mantenimiento de software se ha convertido en uno de los principales objetivos del desarrollo de software a nivel mundial, y nuestra Universidad se hace

partícipe de esta tendencia. Es por ello que en aras de construir un sistema para la generación de reportes con la calidad requerida, este capítulo se ha dedicado a realizar un estudio sobre metodologías de desarrollo de software, así como de herramientas Case y lenguajes de modelado para guiar el proceso de desarrollo. 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, ventajas y desventajas 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

2.1. Introducción

Este capítulo constituye la propuesta de solución para estandarizar el proceso de generación de reportes en la Universidad. Para describir la solución se realiza el modelo de dominio, así como 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.

2.1. 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 conceptual no es una descripción de los componentes del software; representa los conceptos en el dominio del problema en el mundo real. Este constituye el artefacto más importante a crear durante el análisis orientado a objetos. (Larman, 1999) Utilizando UML, un modelo de dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación.

Los modelos de dominio pueden mostrar:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales. (Sánchez Ríos, 2007)

Las clases del dominio 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)

2.1.1. ¿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 implementar un 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 Universidad, 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.

2.2. 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:

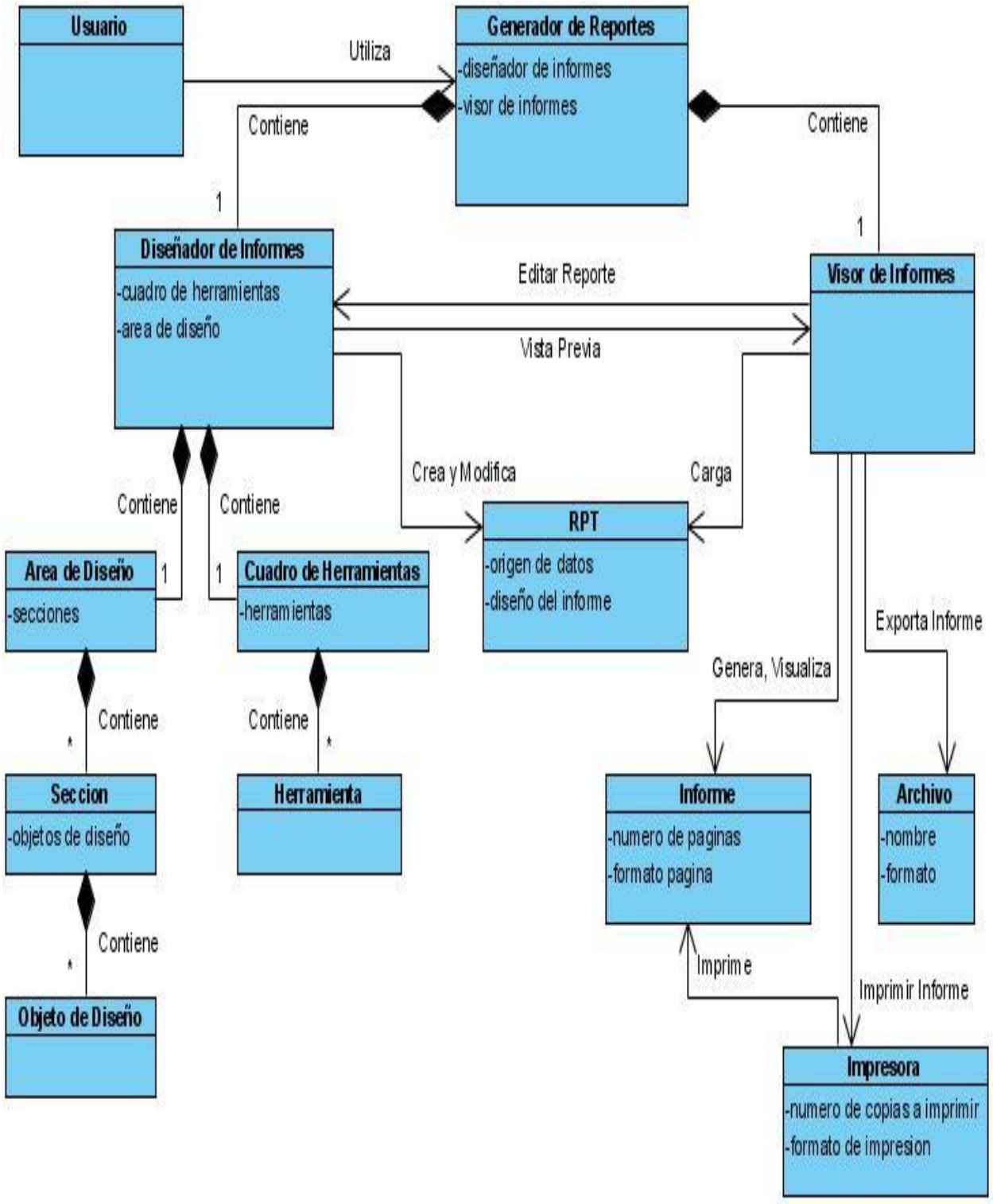


Figura 2.1. Modelo de Dominio

2.3. 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 consenso 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. (Rumbaugh, y otros, 2000)

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

2.4. Modelo del sistema

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

2.4.1. 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. (Rumbaugh, y otros, 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 los 6 meses y los 3 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

2.4.1.1. 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 Crear reporte

R. 1.1 Definir el formato del reporte

R. 2 Modificar diseño reporte

R. 3 Obtener una vista previa del reporte

R. 4 Guardar el diseño del reporte

R. 5 Cambiar el origen de datos

R. 5.1 Mostrar los datos dinámicos presentes en el reporte

R. 5.2 Definir la fuente de datos

R 5.3 Conectar con la fuente de datos

R 5.4 Mostrar los datos disponibles en la fuente seleccionada

R 5.6 Seleccionar que datos de la fuente formarán parte del reporte

R 5.7 Eliminar datos del reporte

R 6 Editar una consulta

R 7 Adicionar un objeto de diseño

R 8 Eliminar objeto de diseño

R 9 Modificar objeto de diseño

R 10 Insertar expresión

R 11 Modificar expresión

R 12 Eliminar expresión

R 13 Insertar Sección

R 14 Modificar Sección

R 15 Eliminar Sección

R 16 Visualizar Reporte

R 17 Imprimir reporte

R 17.1 Configurar las opciones de impresión

R 18 Exportar reporte

R 18.1 Definir el nombre del archivo a exportar

R 18.2 Definir la ubicación del archivo a exportar

R 18.3 Definir el formato en el que se desea exportar

2.4.1.2. 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.

2.4.2. 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. (Rumbaugh, y otros, 2004)

2.4.2.1. 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.

2.4.2.2. Casos de uso del sistema

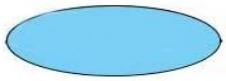


Figura 2.3 Estereotipo de Caso de uso del sistema

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. (Rumbaugh, y otros, 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:

Tabla 2.2 Resumen del caso de uso 1

CU-1	Gestionar diseño del reporte
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario decide crear un reporte o modificar el diseño de uno existente. Permite además al usuario obtener una vista previa del diseño del reporte.
Referencia	R.1, R. 1.1, R. 2, R. 3, R. 4

Tabla 2.3 Resumen del caso de uso 2

CU-2	Cambiar origen de datos
Actor	Usuario

Descripción	El caso de uso se inicia cuando el usuario decide cambiar los datos dinámicos que presentará el reporte.
Referencia	R. 5, R. 5.1, R. 5.2, R. 5.3, R. 5.4, R. 5.5, R. 5.6, R 5.7, R. 6

Tabla 2.4 Resumen del caso de uso 3

CU-3	Gestionar objeto de diseño
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar un objeto de diseño.
Referencia	R. 7, R. 8, R. 9

Tabla 2.5 Resumen del caso de uso 4

CU-4	Gestionar expresión
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar una expresión.
Referencia	R. 10, R. 11, R. 12

Tabla 2.6 Resumen del caso de uso 5

CU-5	Gestionar Sección
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar una sección.
Referencia	R. 13, R. 14, R. 15

Tabla 2.7 Resumen del caso de uso 6

CU-6	Visualizar Reporte
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario decide visualizar el contenido de un reporte. Para luego poderlo imprimir o exportar.
Referencia	R. 16, R. 17, R. 17.1, R. 18, R. 18.1, R.18.2, R18.3

2.4.2.3. Diagrama de Casos de uso del sistema.

El Diagrama de Casos de uso describe cómo interactúan los actores y los casos de uso y como se relacionan entre sí los casos de uso. (Rumbaugh, y otros, 2004). Es un modelo de las funciones del sistema y su entorno, y sirve como un contrato entre el cliente y los desarrolladores. El modelo de casos de uso del sistema es usado como una entrada fundamental a las actividades en el análisis, diseño y las pruebas. (Rational Software Corporation, 2003)

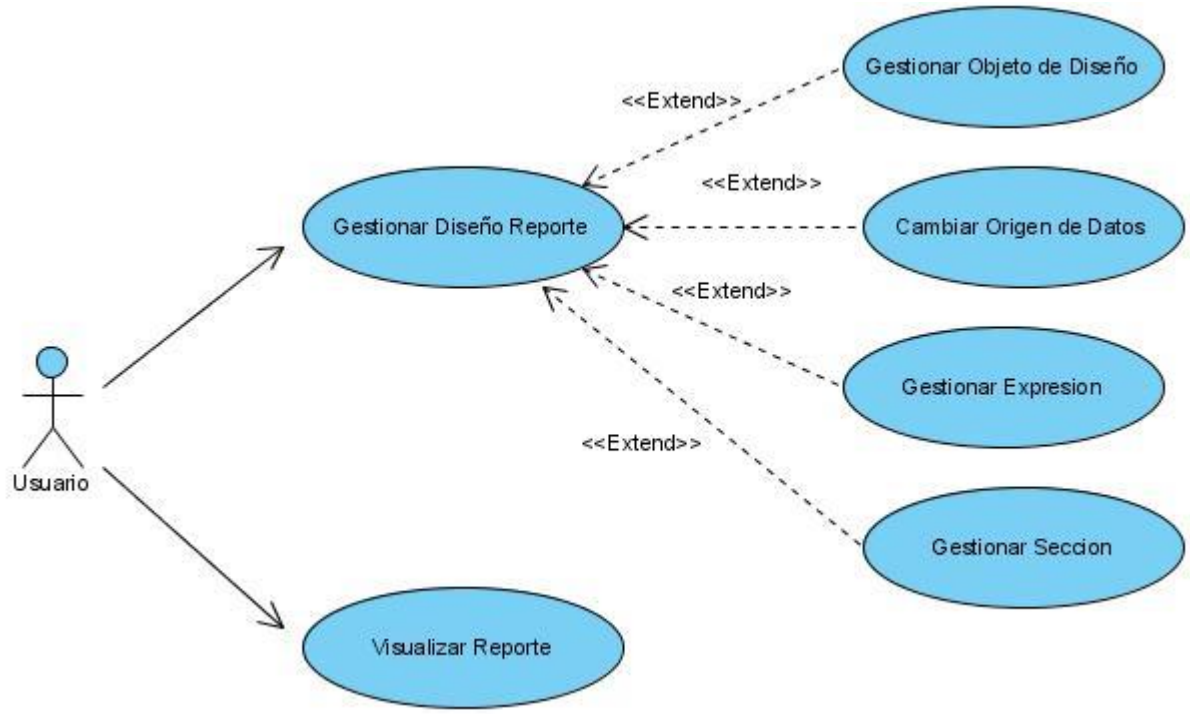


Figura 2.4 Diagrama de Casos de uso del sistema.

Para estructurar el diagrama de casos de uso del sistema se tuvieron en cuenta los siguientes patrones de casos de uso: CRUD completo, Servicio Opcional: Adición.

2.4.2.4. **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. (Rational Software Corporation, 2003)

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 diseño del reporte

Caso de Uso:	Gestionar diseño del reporte
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide crear un reporte o modificar uno existente.
Precondiciones:	Para modificar un reporte, deben existir reportes creados.
Referencias:	R.1, R. 1.1, R. 2, R. 3, R. 4
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción crear o abrir reporte Si es: a) Crear reporte: Ver Sección Crear reporte b) Abrir Reporte: Ver Sección Modificar diseño reporte	
Sección: Crear reporte	
1. El usuario selecciona la opción Crear reporte	2. El sistema muestra una interfaz solicitando:

	• Nombre • Ubicación • Plantilla a partir de la cual se creará el reporte (en blanco o sobre un reporte existente)
3. El usuario selecciona la opción reporte en blanco, presiona el botón aceptar. Si el usuario selecciona el botón Cancelar ver flujo alterno 2	4. El sistema muestra el reporte para su edición.
5. El usuario diseña el reporte y guarda los cambios efectuados, si desea obtener una vista previa del reporte: Ver sección Vista Previa	
Flujos Alternos	
Flujo Alterno1	
3.1. El usuario selecciona la opción crear a partir de un reporte existente	3.2. El sistema muestra una ventana para especificar la ubicación del reporte existente.
3.3. Si el usuario selecciona la opción Aceptar Volver al paso 4 del flujo normal de eventos Si el usuario selecciona la opción Cancelar Ver Flujo Alterno 2.	
Flujo Alterno 2	
	4.2. El sistema cierra la interfaz de creación del reporte
Sección: Modificar diseño reportes	

1. El usuario selecciona la opción Abrir Reporte	2. El sistema muestra una ventana para buscar el reporte.
3. El usuario busca y selecciona el reporte	4. El sistema muestra el reporte para su edición
5. El usuario diseña el reporte y guarda los cambios efectuados, si desea obtener una vista previa del reporte: Ver sección Vista Previa	
Sección Vista Previa	
1. El usuario selecciona la opción Vista Previa	2. El sistema muestra una vista previa del reporte.
Prototipo de Interfaz Sección: Crear Reporte	
Ver Anexos Figura 1 y Figura 2.	
Prototipo de Interfaz Sección Modificar Reportes	
Ver Anexos Figura 3	
Poscondiciones:	Se obtiene el reporte con el diseño deseado

Descripción detallada del CU Cambiar origen de datos

Caso de Uso:	Cambiar Origen de Datos
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide cambiar los datos dinámicos que presentará el reporte.
Precondiciones:	Tiene que existir un reporte en edición.
Referencias:	R. 5, R. 5.1, R. 5.2, R. 5.3, R. 5.4, R. 5.5, R. 5.6, R 5.7, R. 6
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción Cambiar	2. El sistema muestra una interfaz para

origen de Datos	modificar los datos dinámicos del reporte
3. Si el usuario desea : a) Adicionar datos dinámicos al reporte: Ver Sección Adicionar datos dinámicos. b) Eliminar datos dinámicos del reporte: Ver Sección Eliminar datos. c) Editar consulta: Ver Sección Editar Consulta.	
Sección Adicionar Datos Dinámicos	
1. El usuario selecciona el tipo de fuente de datos, selecciona la opción conectarse a una fuente de datos del tipo especificado.	2. El sistema muestra una interfaz solicitando: • Nombre de la fuente de datos • Ubicación de la fuente de datos.
3. El usuario especifica los datos requeridos, selecciona la fuente de datos	4. El sistema muestra los datos disponibles en la fuente seleccionada.
5. El usuario selecciona que datos de la fuente formarán parte del reporte y los adiciona a la sección Datos del Reporte.	6. El sistema muestra los datos en la sección Datos del Reporte
7. Si el usuario selecciona la opción aceptar	8. El sistema cierra la interfaz de modificación de datos y refleja los cambios en la interfaz de diseño.

Flujo Alterno	
7.1. Si el usuario selecciona la opción Cancelar	7.2. El sistema cierra la interfaz de modificación de datos, no se adiciona ningún dato dinámico al reporte
Sección Eliminar Datos	
1. El usuario selecciona los datos que va a eliminar del reporte y elige la opción Eliminar	2. El sistema elimina los datos de la sección datos del reporte.
3. Si el usuario selecciona la opción Aceptar	4. El sistema cierra la interfaz y refleja los cambios en la interfaz de diseño.
Flujo Alterno	
3.1. Si el usuario selecciona la opción Cancelar	3.2. El sistema cierra la interfaz de modificación de datos, los datos del reporte no sufren ninguna modificación.
Sección Editar Consulta	
1. El usuario selecciona la opción editar consulta	2. El sistema muestra una interfaz para editar la consulta
3. El usuario modifica la consulta, selecciona la opción chequear.	4. El sistema muestra un mensaje indicando la validez de la consulta.
5. El usuario selecciona la opción Aceptar	6. El sistema cierra la interfaz de edición de la consulta.
Flujo Alterno	
3.1. Si el usuario presiona la opción Cancelar	3.2. El sistema cierra la interfaz de edición de la consulta, sin realizar

	modificaciones.
Prototipo de Interfaz	
Prototipo de Interfaz Seccionar Adicionar y Eliminar datos dinámicos	
Ver Anexos Figura 4	
Prototipo de Interfaz Sección Editar Consulta	
Ver Anexos Figura 5	
Poscondiciones:	Quedan definidos los datos dinámicos del reporte en edición.

Descripción detallada del CU

Caso de Uso:	Gestionar Objeto de Diseño
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar un objeto de diseño.
Precondiciones:	Tiene que existir un reporte en edición
Referencias:	R. 7, R. 8, R. 9
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona el objeto de diseño que desea insertar, modificar o eliminar. Si es: a) Insertar objeto de diseño: Ver Sección Insertar objeto de diseño. b) Modificar objeto de diseño: Ver Sección Modificar objeto de diseño. c) Eliminar objeto de diseño: Ver Sección	

Eliminar objeto de diseño	
Sección: Insertar objeto de diseño	
1. El usuario selecciona en la barra de herramientas el tipo de objeto que desea insertar y lo coloca en el área de diseño.	2. El sistema crea un nuevo objeto del tipo seleccionado en la posición indicada por el usuario.
Flujo Alterno	
	2.1. El sistema muestra una ventana de configuración.
2.2. El usuario configura el objeto.	Regresa al paso 2 del flujo normal de eventos.
Sección Modificar objeto de diseño	
1. El usuario selecciona el objeto que desea modificar, elige la opción Ver Propiedades.	2. El sistema muestra en la Ventana de Propiedades las propiedades del objeto seleccionado.
3. El usuario modifica las propiedades del objeto seleccionado	4. El sistema refleja los cambios efectuados.
Flujo Alterno	
1.1. Si el objeto presenta propiedades adicionales el usuario elige la opción Otras Configuraciones	1.2. El sistema muestra una ventana de configuración.
Regresa al paso 3 del flujo normal de eventos	
Sección Eliminar objeto de diseño	
1. El usuario selecciona el objeto que desea eliminar y elige la opción eliminar	2. El sistema muestra una ventana verificando si el usuario realmente desea eliminar el objeto de diseño.

3. El usuario selecciona la opción Aceptar.	4. El sistema cierra la ventana, elimina el objeto de diseño.
Flujo Alterno	
3.1. El usuario selecciona la opción Cancelar	3.2. El sistema cierra la ventana, sin eliminar el objeto de diseño.
Prototipo de Interfaz	
Ver Anexos Figura 6	
Poscondiciones	Se inserta, se modifica o se elimina un objeto de diseño.

Descripción detallada del CU Gestionar Expresión

Caso de Uso:	Gestionar expresión
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar una expresión.
Precondiciones:	Tiene que existir un reporte en edición
Referencias:	R. 10, R. 11, R. 12
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción insertar, modificar o eliminar una expresión. Si es: a) Insertar Expresión: Ver Sección Insertar Expresión. b) Modificar Expresión: Ver Sección Modificar Expresión. c) Eliminar Expresión: Ver Sección Eliminar Expresión	
Sección Insertar Expresión	

1. El usuario elige la opción insertar expresión	2. El sistema muestra una interfaz para crear la expresión
3. El usuario crea la expresión, elige la opción chequear.	4. El sistema muestra una ventana indicando la validez de la expresión
5. El usuario selecciona la opción aceptar	6. El sistema cierra la interfaz, mostrando la expresión en el formulario.
Flujo Alterno	
5.1. El usuario selecciona la opción Cancelar	5.2. El sistema cierra la interfaz.
Sección Modificar Expresión	
1. El usuario selecciona la expresión que desea modificar, elige la opción modificar expresión.	2. El sistema muestra una interfaz para modificar la expresión.
3. El usuario modifica la expresión, elige la opción chequear.	4. El sistema muestra una ventana indicando la validez de la expresión
5. El usuario selecciona la opción aceptar	6. El sistema cierra la interfaz, mostrando la expresión modificada en el formulario.
Flujo Alterno	
5.1. El usuario selecciona la opción Cancelar	5.2. El sistema cierra la interfaz.
Sección Eliminar Expresión	
1. El sistema selecciona la expresión, elige la opción eliminar.	2. El sistema muestra una ventana para confirmar si realmente se desea eliminar esta expresión.
3. El usuario selecciona la opción Aceptar	4. El sistema elimina la expresión del

	formulario.
Flujo Alterno	
3.1. El usuario selecciona la opción Cancelar	3.2. El sistema cierra la ventana sin eliminar la expresión del formulario
Prototipo de Interfaz	
Ver Anexos Figura 7	
Poscondiciones	Se inserta, se modifica o se elimina una expresión.

Descripción detallada del CU Gestionar Sección

Caso de Uso:	Gestionar Sección
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide insertar, modificar o eliminar una sección.
Precondiciones:	Tiene que existir un reporte en edición
Referencias:	R. 13, R. 14, R. 15
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción gestionar secciones Si desea: a) Insertar Sección: Ver Sección Insertar Sección b) Modificar Sección: Ver Sección Modificar Sección. c) Eliminar Sección : Ver Sección Eliminar Sección	

Sección Insertar Sección	
1. El usuario elige el tipo de sección que desea insertar, elige la opción Insertar	2. El sistema crea la nueva sección, la selecciona y muestra sus propiedades.
3. El usuario selecciona la opción aceptar	4. El sistema cierra la interfaz, mostrando la sección en el área de diseño
Flujo Alterno	
3.1. El usuario selecciona la opción Cancelar	3.2. El sistema cierra la interfaz.
Sección Modificar Sección	
1. El usuario selecciona la sección que desea modificar	2. El sistema muestra las propiedades de la sección seleccionada.
3. El usuario modifica las propiedades deseadas, selecciona la opción aceptar.	4. El sistema cierra la interfaz, mostrando la sección modificada en el formulario.
Flujo Alterno	
3.1. El usuario selecciona la opción cancelar	3.2. El sistema cierra la interfaz.
Sección Eliminar Sección	
1. El sistema selecciona la sección que desea eliminar, elige la opción eliminar.	2. El sistema muestra una ventana para confirmar si realmente se desea eliminar esta expresión.
3. El usuario selecciona la opción Aceptar	4. El sistema elimina la sección del reporte
Flujo Alterno	
3.1. El usuario selecciona la opción Cancelar	3.2. El sistema cierra la ventana sin

	eliminar la expresión del formulario
Prototipo de Interfaz de usuario	
Ver Anexos Figura 8	
Poscondiciones	Se inserta, se modifica o se elimina una sección

Descripción detallada del CU Visualizar Reporte

Caso de Uso:	Visualizar Reporte
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario decide visualizar el contenido de un reporte.
Precondiciones:	Tiene que existir un reporte previamente diseñado.
Referencias:	R. 16, R. 17, R. 17.1, R. 18, R. 18.1, R.18.2, R18.3
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción visualizar reporte.	2. El sistema muestra una ventana para que el usuario especifique el reporte que desea visualizar
3. El usuario especifica el reporte que desea visualizar	4. El sistema carga el diseño del reporte y genera y visualiza el informe.
5. Si el usuario desea: a) Imprimir el reporte: Ver sección Imprimir Reporte b) Exportar el reporte: Ver sección Exportar Reporte	

Sección Imprimir Reporte	
1. El usuario selecciona la opción de imprimir el reporte	2. El sistema muestra una pantalla con las opciones de impresión.
3. El usuario configura la impresión.	4. El sistema procede a la impresión del reporte.
Sección Exportar Reporte	
1. El usuario selecciona la opción de exportar el reporte	2. El sistema muestra una ventana para especificar los siguientes datos: • Formato a exportar • Nombre del archivo • Ubicación del archivo.
3. El usuario especifica los datos solicitados y manda a exportar.	4. El sistema exporta el informe.
Prototipo de Interfaz	
Ver Anexos Figura 9	
Poscondiciones	Se visualiza un reporte que opcionalmente se puede imprimir y/o exportar

2.5. Conclusiones parciales

Las técnicas de requisitos empleadas permitieron determinar las funcionalidades principales del sistema a construir, de estas funcionalidades especificadas se obtuvieron seis casos de uso, los cuales van a permitir la construcción de un sistema que estandarizará el proceso de generación de reportes en la Universidad.

CAPITULO 3. ÁNALISIS Y DISEÑO DEL SISTEMA

3.1. Introducción

El presente capítulo contiene lo referente al análisis y diseño, realizado como parte de la propuesta de solución. Como parte de esta solución se desarrollan los diagramas de clases y de interacción del análisis y diseño. Conjuntamente con esto se exponen los patrones de diseño empleados en la solución.

3.2. Análisis

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. En el análisis podemos estructurar los requisitos de manera que nos facilite su comprensión, su preparación, su modificación, y en general su mantenimiento (Rumbaugh, et al., 2004)

3.2.1. Modelo de análisis

El lenguaje que utilizamos en el análisis se basa en un modelo de objetos conceptual que llamamos modelo de análisis. El modelo de análisis nos ayuda a estructurara los requisitos y nos 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, obtenemos 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.

3.2.1.1. 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 participantes, 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

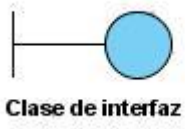


Figura 3.1. Estereotipo de la clase de interfaz.

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 3.2. 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 3.3. 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

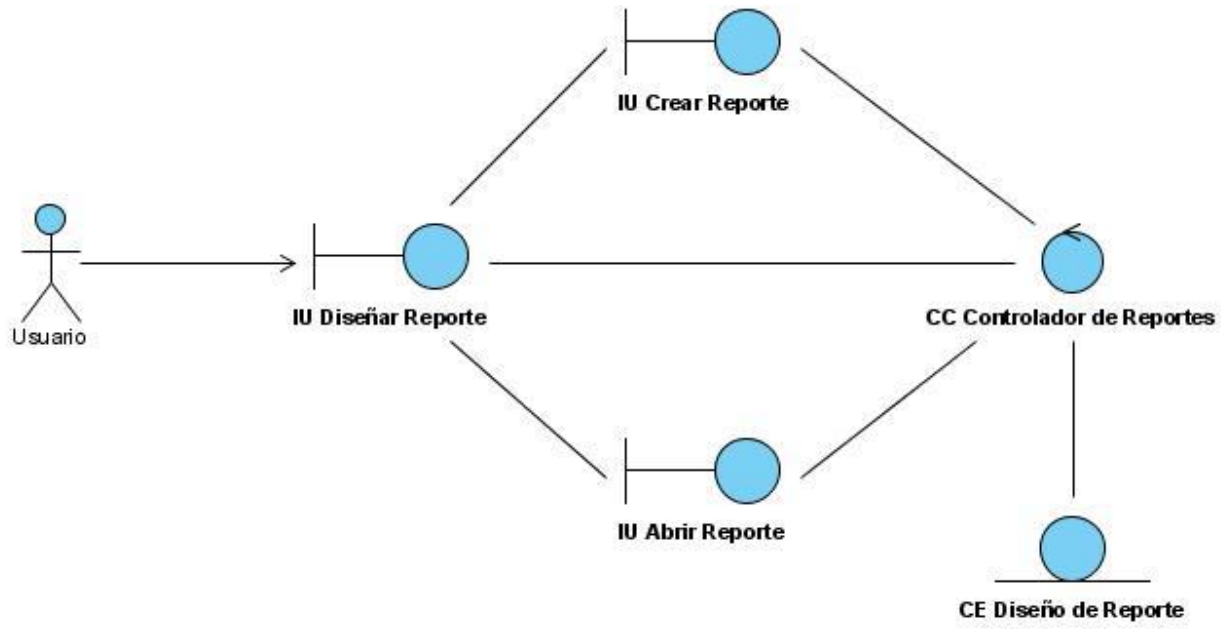


Figura 3.4. Diagrama de clases del análisis del CU Gestionar diseño del reporte

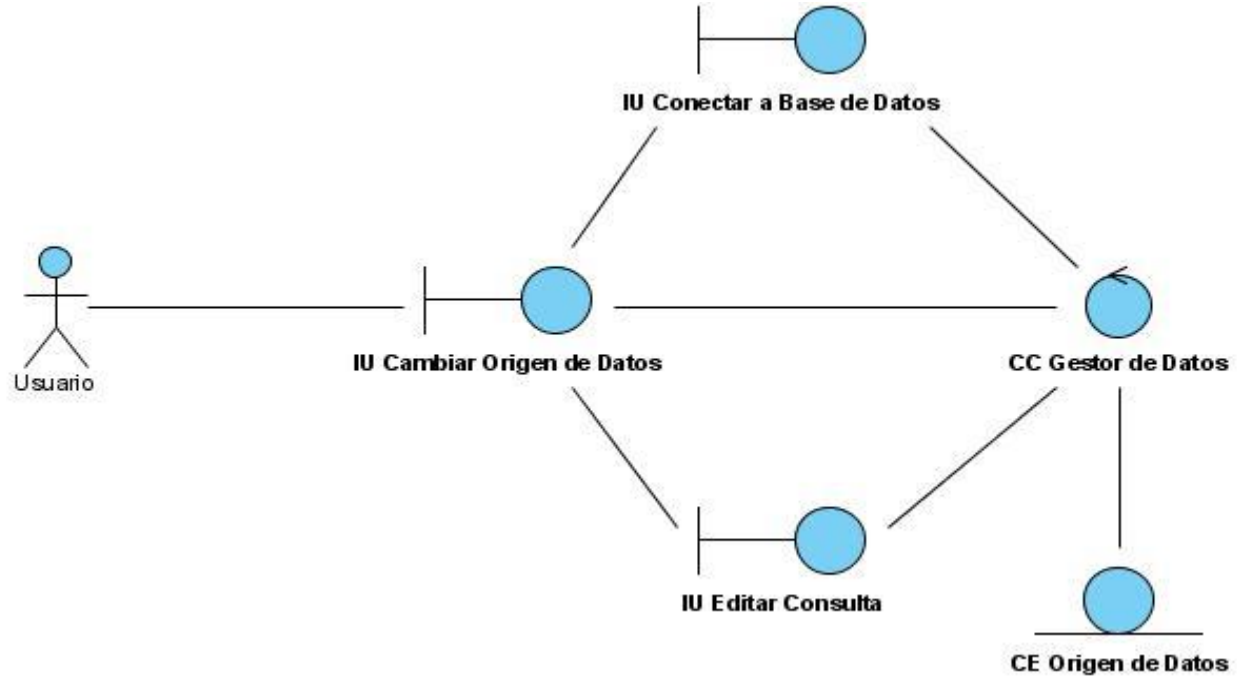


Figura 3.5. Diagrama de clases del análisis del CU Cambiar origen de datos

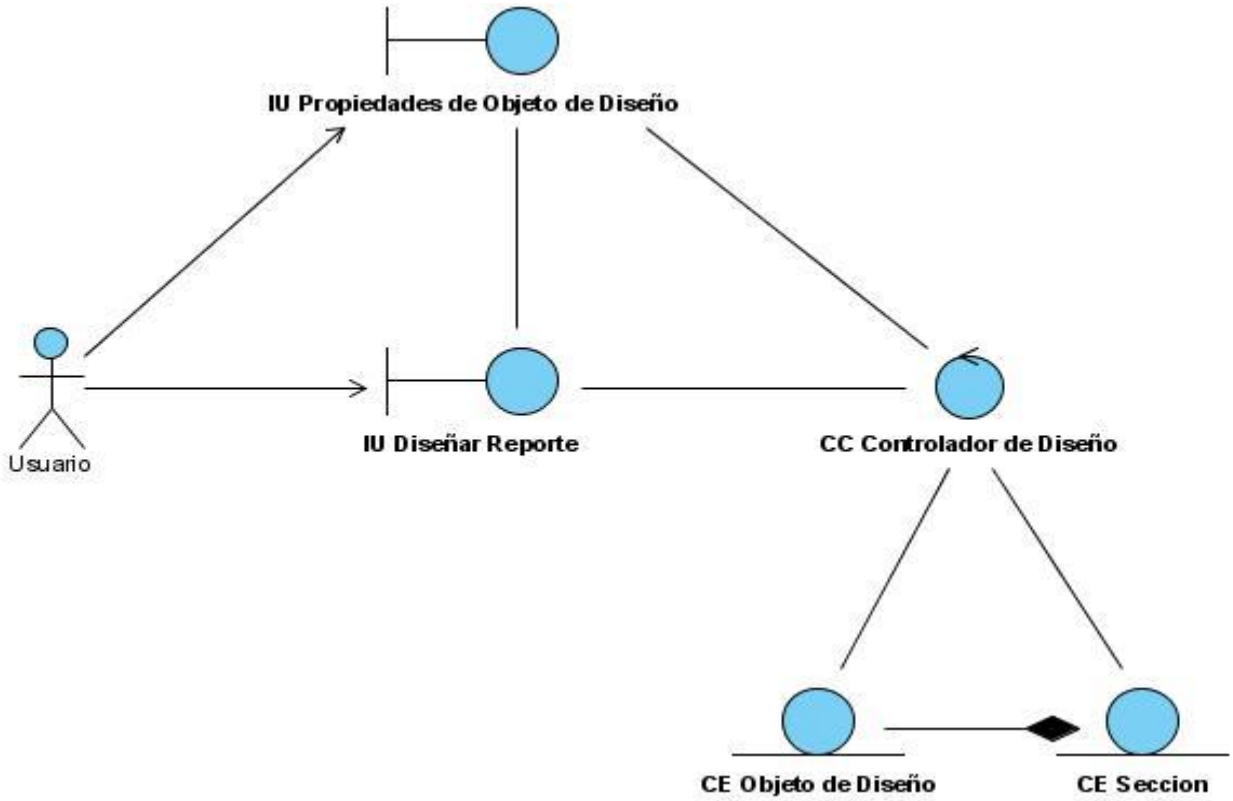


Figura 3.6. Diagrama de clases del análisis del CU Gestionar objeto de diseño

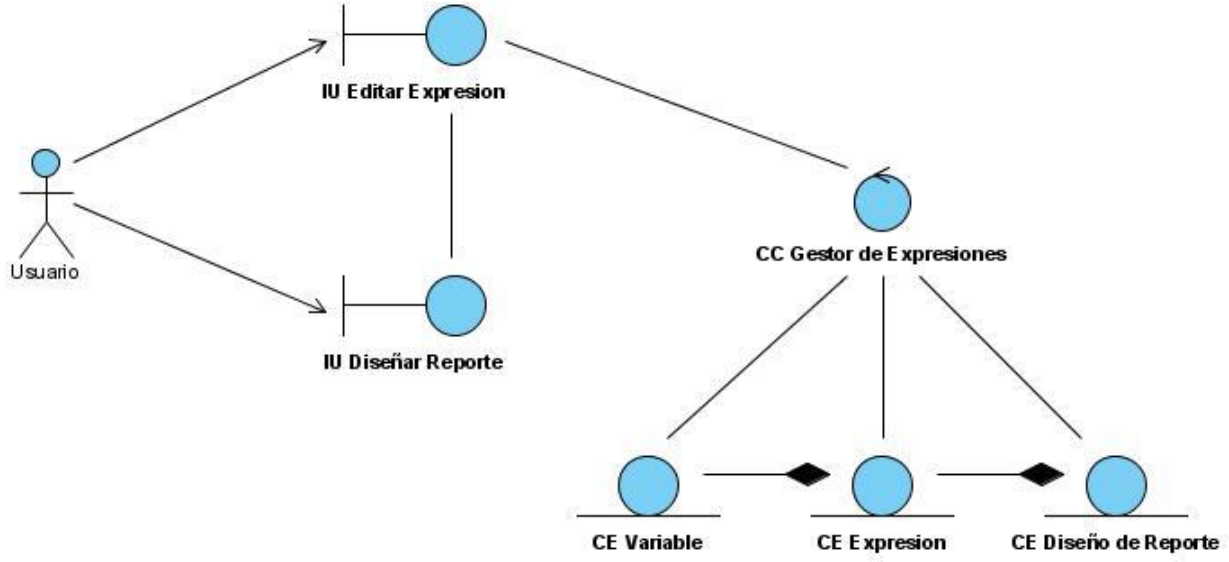


Figura 3.7. Diagrama de clases del análisis del CU Gestionar expresión

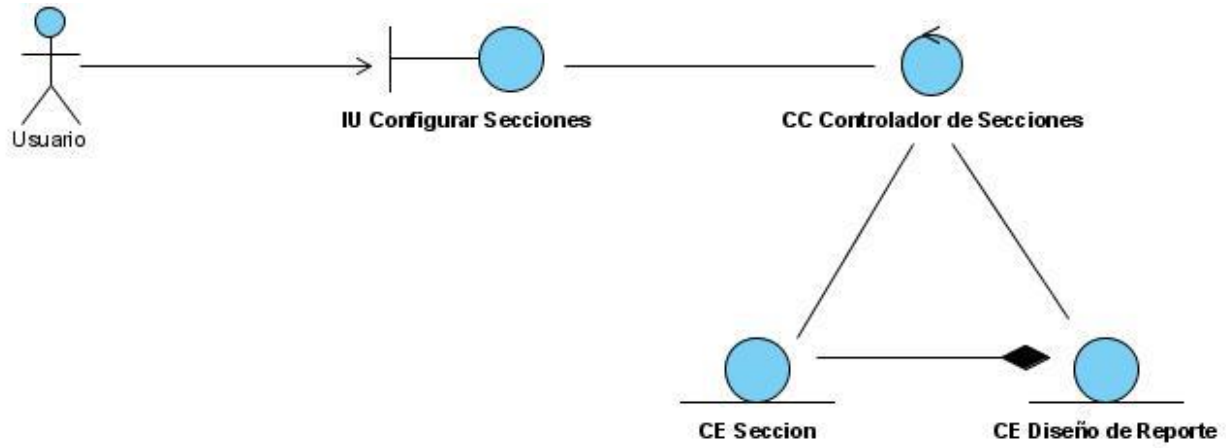


Figura 3.8. Diagrama de clases del análisis del CU Gestionar sección

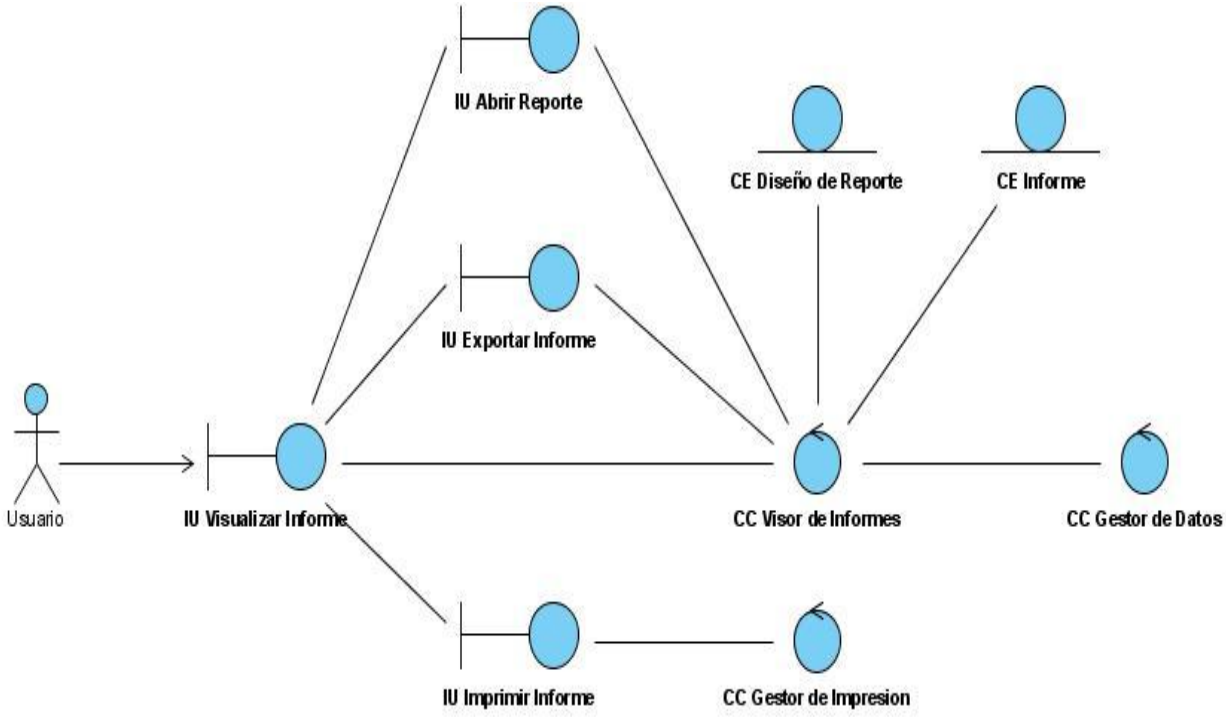


Figura 3.9. Diagrama de clases del análisis del CU Visualizar Reporte

3.2.1.2. 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, et al., 2004) 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 10-23**

3.3. 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, et al., 2004)

3.3.1. 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.

3.3.1.1. 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:

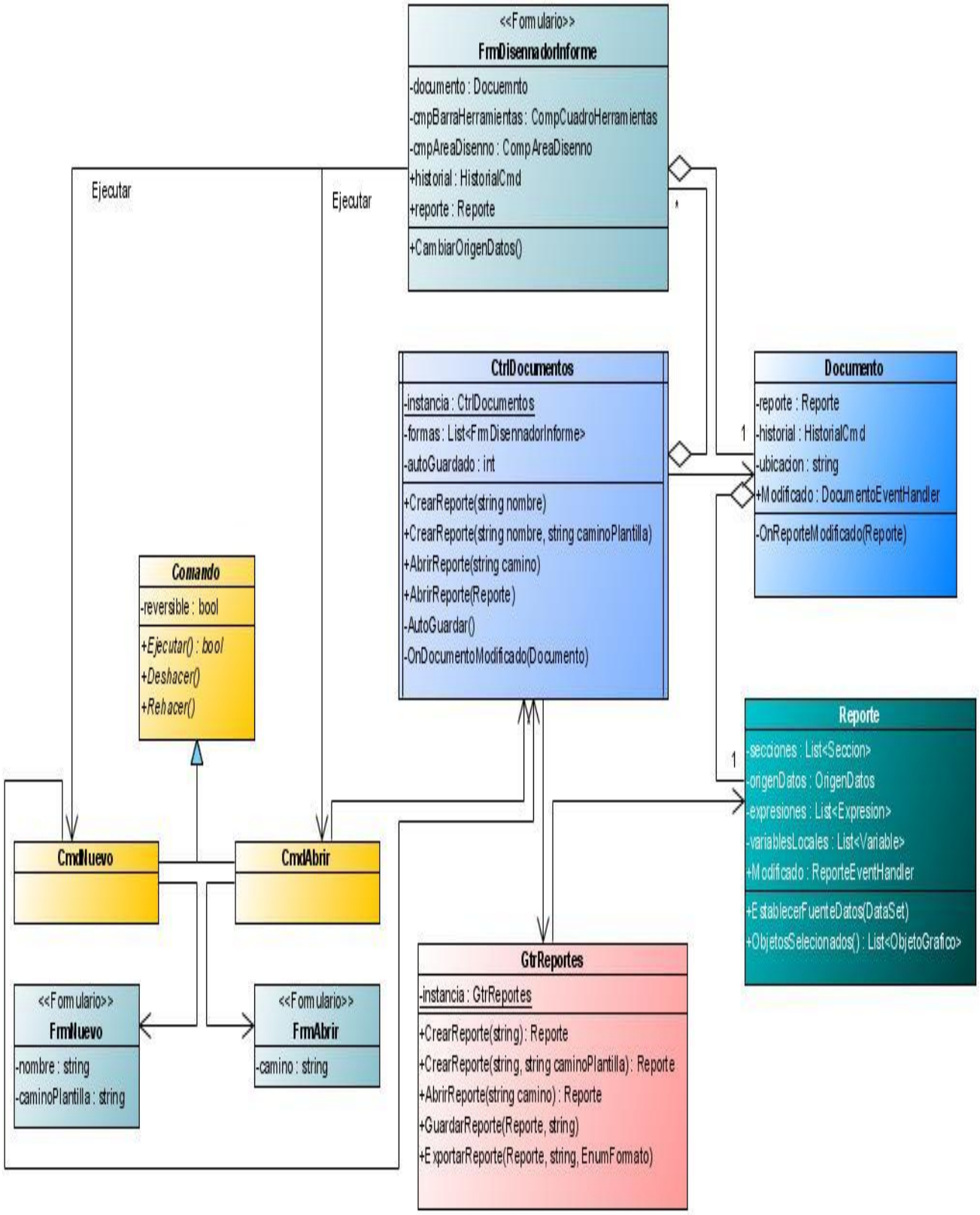


Figura 3.10. Diagrama de clases del diseño CU Gestionar Diseño Reporte

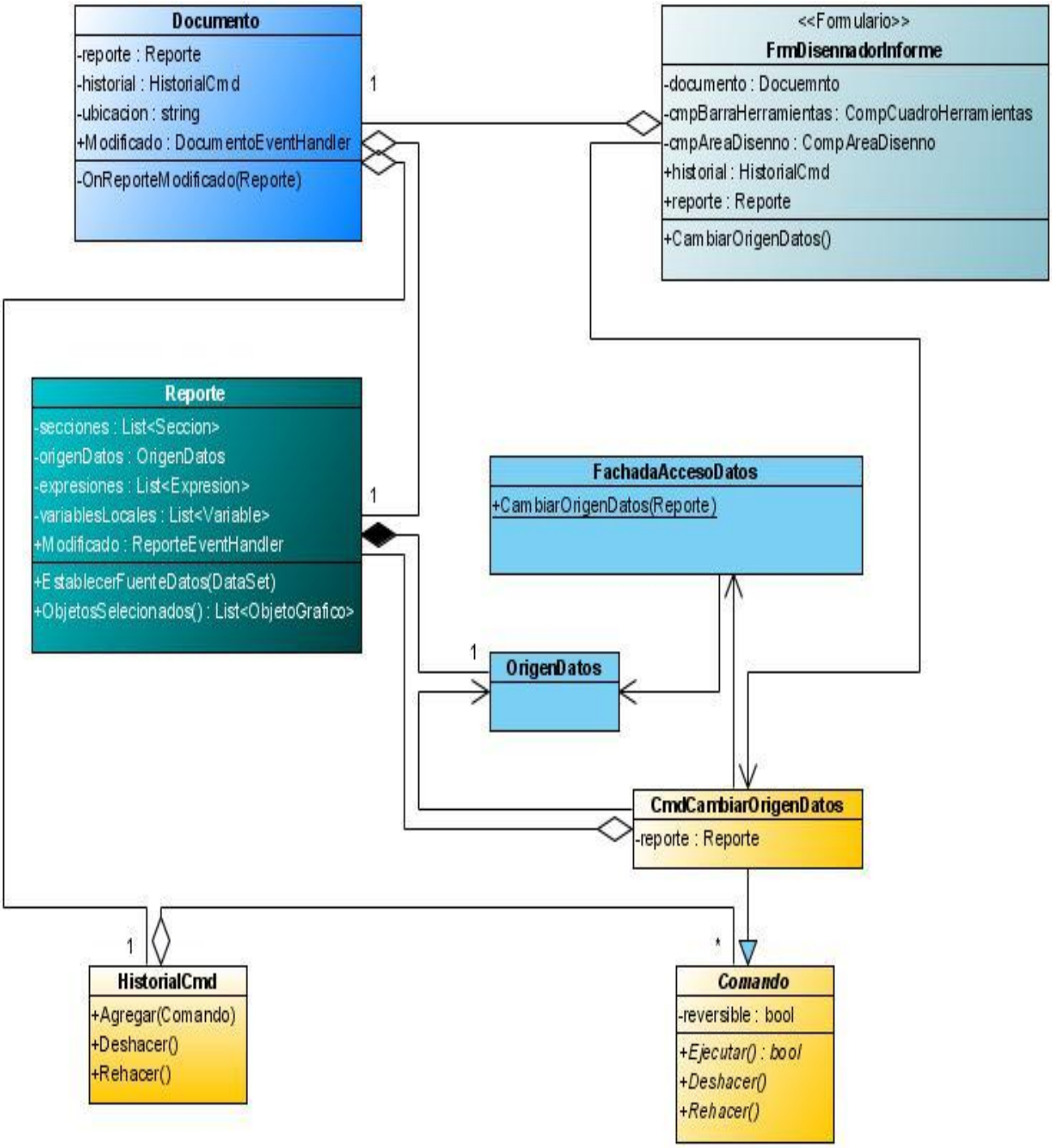


Figura 3.11. Diagrama de clases del diseño CU Cambiar Origen de datos

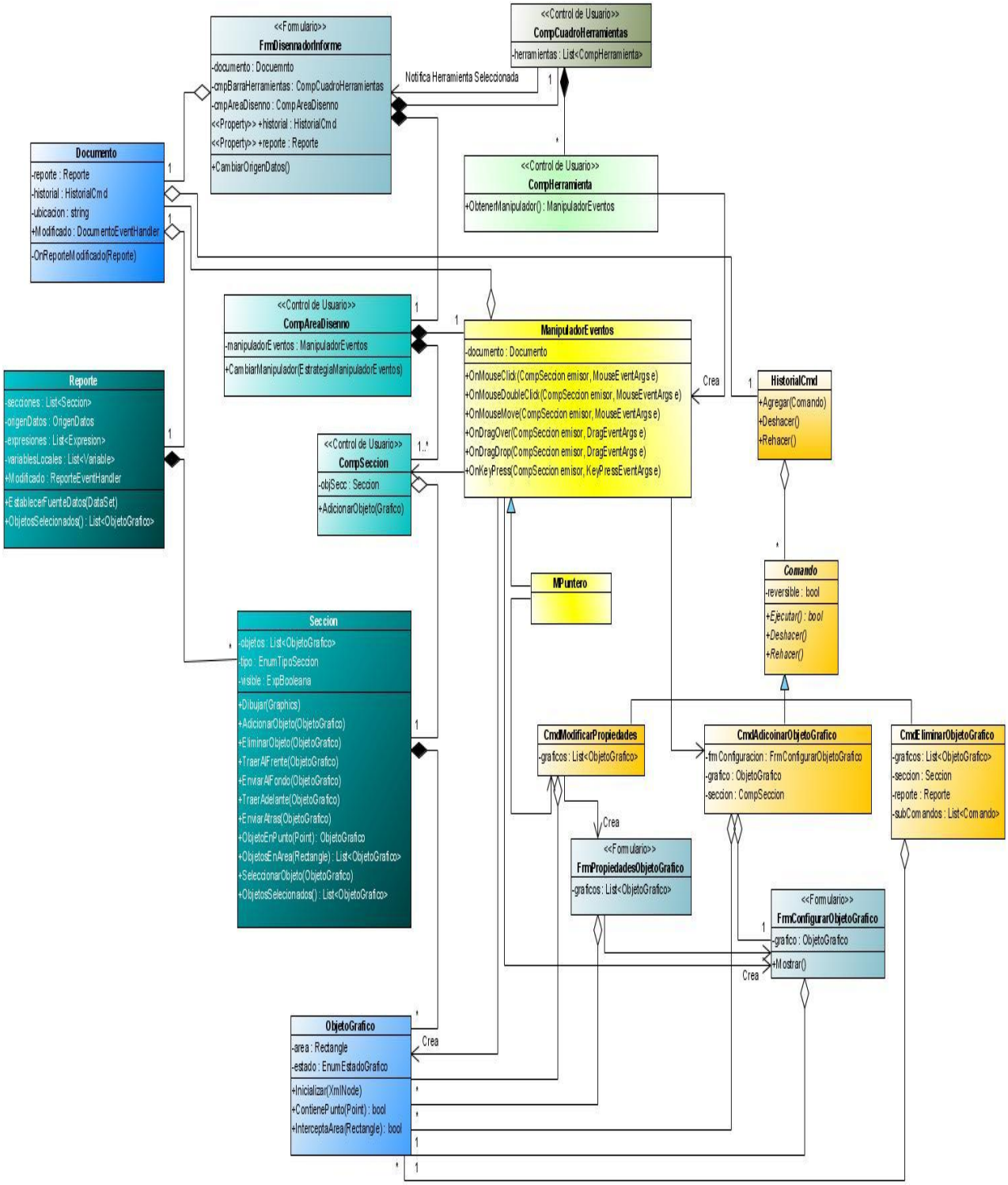


Figura 3.12. Diagrama de clases del diseño CU Gestionar objeto de diseño

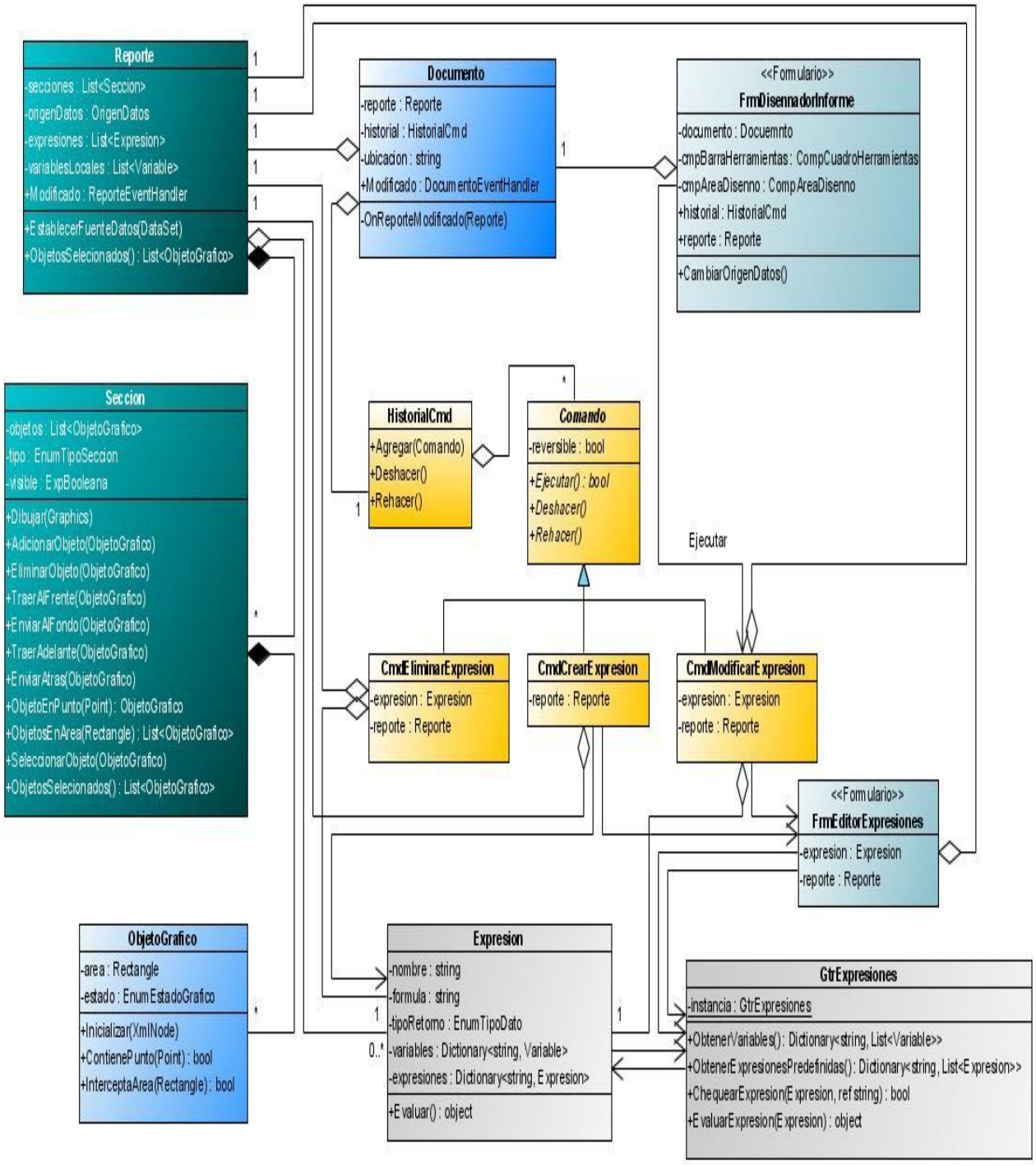


Figura 3.13. Diagrama de clases del diseño CU Gestionar expresión

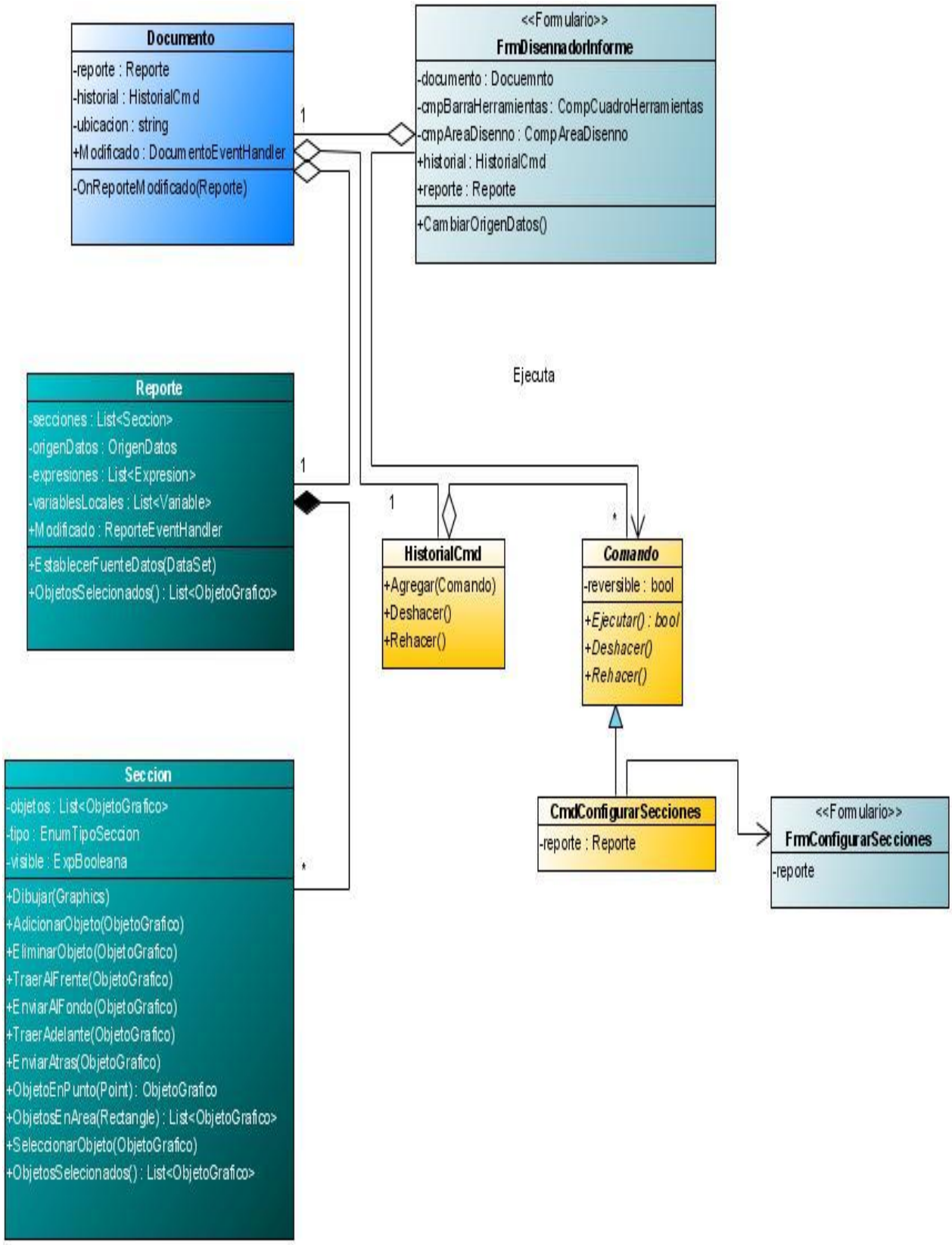

Documento
-reporte : Reporte
-historial : HistorialCmd
-ubicacion : string
+Modificado : DocumentoEventHandler
-OnReporteModificado(Reporte)
<<Formulario>>
FrmDisenradorInforme
-documento : Docuemnto
-cmpBarraHerramientas : CompCuadroHerramientas
-cmpAreaDisenno : CompAreaDisenno
+historial : HistorialCmd
+reporte : Reporte
+CambiarOrigenDatos()
1
Ejecuta
Reporte
-secciones : List<Seccion>
-origenDatos : OrigenDatos
-expresiones : List<Expresion>
-variablesLocales : List<Variable>
+Modificado : ReporteEventHandler
+EstablecerFuenteDatos(DataSet)
+ObjetosSelecionados() : List<ObjetoGrafico>
1
1
*
HistorialCmd
+Agregar(Comando)
+Deshacer()
+Rehacer()
Comando
-reversible : bool
+Ejecutar() : bool
+Deshacer()
+Rehacer()
Seccion
-objetos : List<ObjetoGrafico>
-tipo : EnumTipoSeccion
-visible : ExpBooleana
+Dibujar(Graphics)
+AdicionarObjeto(ObjetoGrafico)
+EliminarObjeto(ObjetoGrafico)
+TraerAlFrente(ObjetoGrafico)
+EnviarAlFondo(ObjetoGrafico)
+TraerAdelante(ObjetoGrafico)
+EnviarAtras(ObjetoGrafico)
+ObjetoEnPunto(Point): ObjetoGrafico
+ObjetosEnArea(Rectangle) : List<ObjetoGrafico>
+SeleccionarObjeto(ObjetoGrafico)
+ObjetosSelecionados() : List<ObjetoGrafico>
*
CmdConfigurarSecciones
-reporte : Reporte
<<Formulario>>
FrmConfigurarSecciones
-reporte

Figura 3.14. Diagrama de clases del diseño CU Gestionar sección

Descripción de las clases del diseño. **Ver Anexos Tablas 2-11**

3.3.1.2. 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 tendremos 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 nuestro centro de atención principal es encontrar secuencias de interacciones detalladas y ordenadas en el tiempo.

Diagramas de secuencia del diseño **Ver índice de figuras 24-37.**

3.4. Patrones empleados en la solución

En la solución se emplearon los siguientes patrones de diseño del Gang of Four (Pandilla de los 4) (GOF): Command, Singleton, Strategy, Facade y Observer

Command (*Comando*): El objetivo de este patrón es encapsular una petición como un objeto, lo que permite parametrizar clientes con diferentes peticiones y brindar soporte a operaciones de deshacer. En el diseño propuesto se hace amplio uso de este patrón para encapsular las acciones requeridas para llevar a cabo peticiones del usuario como editar expresión, adicionar un objeto de diseño, configurar secciones, etc. Luego de ejecutado un comando, este se agrega al historial del documento en edición permitiendo posteriores peticiones de deshacer y rehacer.

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. Este patrón se aplica para las clases CtrlDocumentos, la cual se hará cargo de administrar todos los documentos abiertos, y, la clase GtrReportes las cual será la responsable Gestionar las operaciones de apertura, guardado y exportación de objetos Reporte.

Strategy (*Estrategia*): El patrón Estrategia define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Este patrón permite que el algoritmo varíe independientemente de los clientes que lo usen y es modelado a través de la jerarquía de manipuladores que tiene como base la clase ManipuladorEventos responsable de definir el comportamiento para dar respuesta a algunos eventos del usuario sobre el área de diseño y cuya instancia concreta varía en el sistema en dependencia de la herramienta seleccionada en el cuadro de herramientas.

Facade (*Fachada*): El patrón Fachada provee una interface unificada y de alto nivel a un conjunto de interfaces en un subsistema, haciéndolo más simple de usar. Las ventajas de este patrón fueron aprovechadas para modelar una Fachada para cada uno de los subsistemas que componen el sistema propuesto y así facilitar la comunicación entre ellos propiciando una alta cohesión y un bajo acoplamiento. (Gamma, y otros, 1998)

3.5. Conclusiones parciales

En el presente capítulo utilizando como basamento las características del sistema, que fueron definidas en el capítulo anterior, se realizó el análisis de los casos de uso y el diseño del módulo Diseñador de Reportes, 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 4: ÁNALISIS DE LOS RESULTADOS

4.1. Introducción

Lord Kelvin en una ocasión dijo:

"Cuando pueda medir lo que está diciendo y expresarlo con números, ya conoce algo sobre ello; cuando no pueda medir, cuando no pueda expresar lo que dice con números, su conocimiento es precario y deficiente: puede ser el comienzo del conocimiento, pero en sus pensamientos, apenas está avanzando hacia el escenario de la ciencia."

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.

4.2. 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)

4.2.1. 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:

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

$$n_r = 29 + 15$$

$$n_r = 44$$

Primera Revisión

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

$$Q_i = {}^{35}/_{44}$$

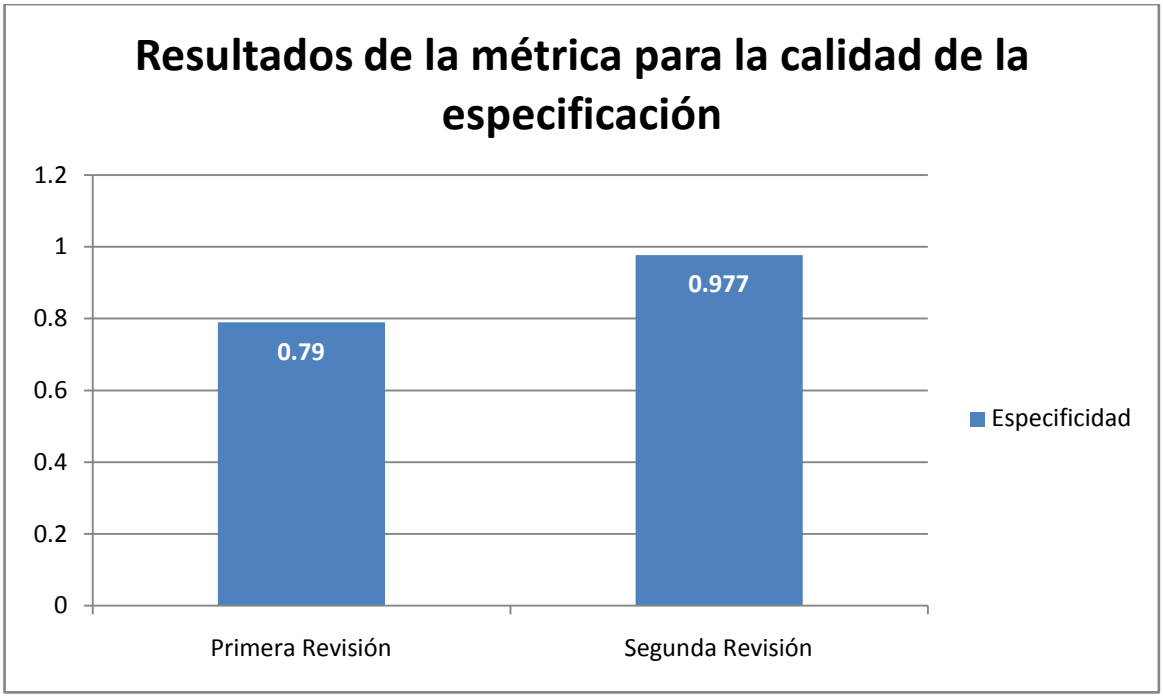
$$Q_i = 0.79$$

Segunda Revisión

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

$$Q_i = {}^{43}/_{44}$$

$$Q_i = 0.977$$



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 43 requisitos de un total de 44, lo que significa la mayoría de los requisitos están especificados de forma clara y sin ambigüedades.

4.2.2. Métricas de Casos de uso

Métrica NOAS[11] /NOS [12]

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 del sistema 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:

[11] Number of Actor Steps- Número de pasos del actor
[12] Number of Steps- Número de pasos

- 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 Diseño del Reporte	10	19	**52.6%**
Cambiar Origen de Datos	8	8	**50%**
Gestionar Objeto de Diseño	7	15	**46.7%**
Visualizar Reporte	6	12	**51%**
Gestionar Expresión	11	22	**50%**
Gestionar Sección	10	19	**52.6%**

Tabla 4.1. 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. En el caso de las acciones del actor, si realiza varias, estas están separadas por comas, lo que constituye un solo paso.

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 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: 13 Número de conceptos que no presentan una descripción detallada: 0 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.	Cantidad de requisitos: 30 Número de requisitos omitidos por caso de

	Métrica 4. Número de casos de uso que tienen requisitos omitidos.	uso: 0 Representa un 0% Número de casos de uso: 7 Cantidad de casos de uso que tienen requisitos omitidos: 0 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: 30 Número de requisitos que no son considerados en ningún caso de uso 0 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: 7 Cantidad de casos de uso que no describen condiciones de excepción relevantes: 2 Representa un 28,6%

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: 7 Cantidad de casos de uso que no han sido clasificados: 0% 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: 7 Cantidad que no tienen una descripción resumida: 0 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: 7 Cantidad que no tienen una descripción extendida: 0 Representa un 0%
		96.83%

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: 7 Cantidad que no representan una interacción observable por un actor: 0

		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: 7 Cantidad que no tienen el flujo de eventos redactado en el lenguaje del usuario: 0 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: 7 Cantidad de casos de uso complejos que no tienen una separación del flujo básico y de flujos alternos: 0 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: 7 Cantidad que tienen un nombre incorrecto: 0 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 Métrica 15. Número de casos de uso en que los requisitos representados no son comprensibles por el usuario	Total de requisitos: 29 Cantidad de requisitos que no son comprensibles por el usuario: 0 Representa: 0% Total de casos de Uso: 7 Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 0 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: 7 Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema : 2 Representa: 28,6%

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: 7 Número de casos de uso que deben ser modificados para mejorar el proceso actual: 2 Representa: 28,6%
		92.85%
Factor de Complejidad		
Factor 17. ¿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: 7. Número de casos de uso que requieren reubicación de manera que faciliten su interpretación:1 Representa: 14.3%
		95, 87%

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 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: 13 Número de conceptos que no presentan una descripción detallada: 0 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. Métrica 4. Número de casos de uso que tienen requisitos omitidos.	Cantidad de requisitos: 29 Número de requisitos omitidos por caso de uso: 0 Representa un 0% Número de casos de uso: 6 Cantidad de casos de uso que tienen requisitos omitidos: 0 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: 29 Número de requisitos que no son considerados en

		ningún caso de uso 0 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: 6 Cantidad de casos de uso que no describen condiciones de excepción relevantes: 1 Representa un 16,6%
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: 6 Cantidad de casos de uso que no han sido clasificados: 0% 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: 6 Cantidad que no tienen una descripción resumida: 0 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	Total de casos de uso: 6 Cantidad que no tienen

	extendida.	una descripción extendida: 0 Representa un 0%
		98%
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:6 Cantidad que no representan una interacción observable por un actor: 0 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: 6 Cantidad que no tienen el flujo de eventos redactado en el lenguaje del usuario: 0 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: 6 Cantidad de casos de uso complejos que no tienen una separación del flujo básico y de flujos alternos: 0 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: 6 Cantidad que tienen un nombre incorrecto 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 Métrica 15. Número de casos de uso en que los requisitos representados no son comprensibles por el usuario	Total de requisitos: 29 Cantidad de requisitos que no son comprensibles por el usuario: 0 Representa: 0% Total de casos de Uso: 6 Número de casos de uso en que los requisitos representados no son comprensibles por el usuario: 0 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: 6 Número de casos de uso que deben ser modificados para

		adecuarlos a la funcionalidad del sistema : 0 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: 6 Número de casos de uso que deben ser modificados para mejorar el proceso actual: 0 Representa: 0%
		100%
Factor de Complejidad		
Factor 17. ¿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: 6. Número de casos de uso que requieren reubicación de manera que faciliten su interpretación. Representa: 0%
		100%

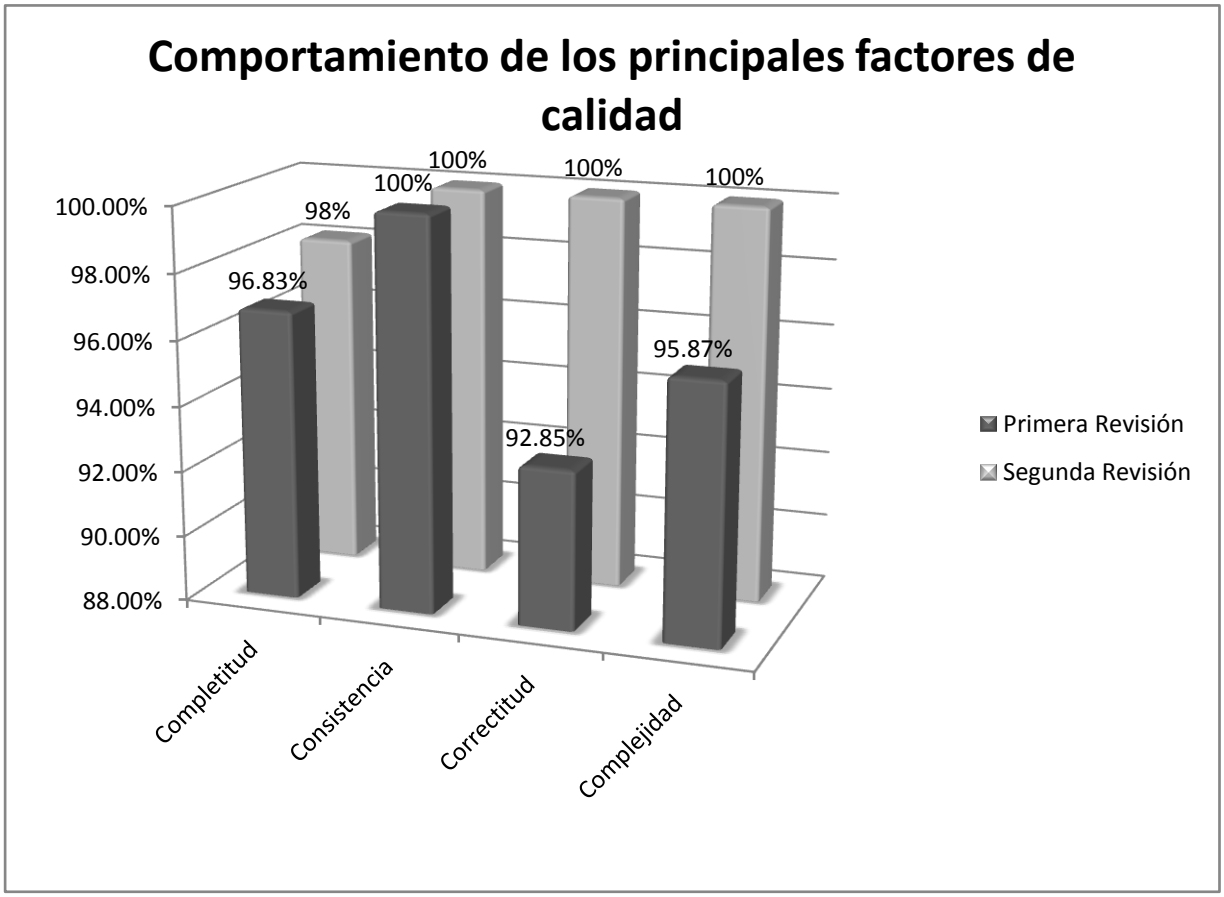


Figura 4.1. Comportamiento de los principales factores de calidad durante las dos revisiones realizadas al DCUS

El gráfico de la figura 4.1 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. 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.

4.2.3. Métricas para el Modelo de Diseño

La serie de métricas CK.

Uno de los conjuntos de métricas más ampliamente referenciados han sido los propuestos por Chidamber y Kemerer. Normalmente conocidas como la serie de métricas CK, los autores han propuesto seis métricas basadas en clases para sistemas OO.

De esas seis métricas se aplicarán las siguientes:

- **Árbol de profundidad de herencia (APH)**

Esta métrica se define como la máxima longitud del nodo a la raíz del árbol. A medida que el APH crece es posible que clases de más bajos niveles heredarán muchos métodos. Esto conlleva dificultades potenciales cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (el APH largo) también conduce a una complejidad de diseño mayor. Por el lado positivo los valores de APH grandes implican un gran número de métodos que se reutilizarán.

Por su parte, algunos autores sugieren un umbral de 6 niveles como indicador de un abuso en la herencia en distintos lenguajes de programación.

Resultado: A partir de los datos obtenidos después de aplicar la métrica APH se obtuvo que los niveles más altos de herencia son de 3, lo cual se encuentra dentro del umbral definido para determinar que el diseño no es complejo, no existe un alto acoplamiento y no es de difícil mantenimiento.

Métricas Propuestas por Lorenz y Kidd

- **Tamaño de clase (TC)**

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 4.2, 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 4.2. Umbrales para TC

Resultado: Esta métrica fue aplicada en la capa de negocio y en la capa de presentación. En la capa de negocios hay un total de 26 clases, y de acuerdo con los umbrales que se presentan en la tabla 4.2 todas las clases se pueden considerar de tamaño pequeño. La capa de presentación tiene un total de 38 clases, todas las clases de acuerdo con los umbrales de la tabla 4.2 se pueden considerar pequeñas. Estos valores demuestran que los indicadores de calidad reutilización, implementación y responsabilidad no se ven afectados.

4.3. 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 sistema.
- Se realizó un análisis profundo del sistema que devino en la obtención de un modelo general independiente del lenguaje de programación a utilizar.
- Se diseñó de manera flexible y robusta el subsistema Diseñador de Informes.
- Se obtuvieron y evaluaron artefactos necesarios según la metodología seleccionada (RUP), para que se le pueda dar continuidad al proceso de desarrollo.

Recomendaciones.

Se recomienda:

- Que se realice el diseño del resto de los módulos del sistema.
- Definir una arquitectura extensible para el sistema.
- Que se continúe con el resto de los flujos de trabajo que propone la metodología RUP (implementación y prueba).
- Incluir a esta solución un modo básico de diseño para aquellos usuarios no tan experimentados en el manejo de herramientas informáticas.
- Una vez terminado el producto, sea puesto en explotación por los proyectos productivos de la Universidad.

Bibliografía.

[En línea] http://www.visual-paradigm.com.

[En línea]

Electronic Computer Glossary. 1995. [En línea] 1995.

Rational Software Corporation. 2003. Rational Unified Process 1.0. s.l. : Corporation, Rational Software, 2003. Vol. 2003.06.00.

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

Bernárdez, B., Durán, A. y Toro, M. 2004. Una propuesta para la verificación de requisitos basada en métricas. *Revista de Procesos y Métricas de las Tecnologías de la Información.* [En línea] Agosto de 2004. [Citado el: 11 de 05 de 2008.] http://www.aemes.org/rpm/descargar.php?volumen=1&numero=2&articulo=2.

Boggs, Wendy y Boggs, Michael. 2002. *UML with Rational Rose.* 2002.

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

Brown, Alan W. 1996. *Component-Based Software Engineering.* s.l. : IEEE Computer Society, Los Alamitos, CA, 1996. pág. 140.

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

Díez, A. 2001. *IRqA y el desarrollo de proyectos: Experiencias Prácticas.I Jornadas de Ingeniería de Requisitos.* España : s.n., 2001.

Escribano, Gerardo Fernándes. 2002. Introducción a Extreme Programming. *http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.* [En línea] 2002.

Ferrer Grau, Xavier. 2005. Tesis de Doctorado Marco de Integración de la usabilidad en el Proceso de Desarrollo de Software. *Universidad Politécnica de Valencia, Facultad de Informática .* [En línea] 2005. [Citado el: 15 de 02 de 08.] http://oa.upm.es/440/01/XAVIER_FERRE_GRAU.PDF.

Gamma, Erich, y otros. 1998. *Design Patterns: Elements of Reusable Object- Oriented Software.* Holland : Addison Wesley, 1998.

García Ávila, Lourdes. 2000. *Modelo de evaluación de la calidad para el análisis y diseño orientado a objetos de sistemas informáticos.* Universidad Central de las Villas : Tesis de Doctorado, 2000.

García Ávila, Lourdes y Fernández Sánchez, Leidy. 2007. Procedimiento para el desarrollo del proceso de ingeniería de requisitos en un proyecto software (PROCIR). [En línea] 2007. [Citado el: 15 de 3 de 2008.] http://www.informaticahabana.com/evento_virtual.

García Rubio, Félix Oscar y Bravo Santos, Crescencio. 2007. Metodologías de Desarrollo de Softaware. [En línea] 11 de 12 de 2007. http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/tema9_2xh.pdf.

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

Ingeniería de Software y su relación con las Herramientas Case. [En línea] [Citado el: 14 de 1 de 2008.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rea_c_ji/capitulo2.pdf.

Jacobson, Ivar, Rumbaugh, James y Booch, Grady. 2004. *El Proceso Unificado de Desarrollo.* La Habana : Félix Varela, 2004.

Jacobson, Ivar, y otros. 1992. *Object-Oriented Software Engineering—A Use Case Driven Approach.* Wokingham : Addison-Wesley, 1992. pág. 582.

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

José Escalona, María y Koch, Nora. 2002. Ingeniería de Requisitos en Aplicaciones para la WEb. Un estudio comparativo. *Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla.* [En línea] 12 de 2002. [Citado el: 24 de 3 de 2008.] http://www.lsi.us.es/docs/informes/LSI-2002-4.pdf.

Joyanes Aguilar, Dr. Luis. Mayo 2000. *Prólogo a la edición en español del Lenguaje Unificado de Modelado.* Madrid : s.n., Mayo 2000.

Joyanes, Luis. 1997. *Cibersociedad: Los retos sociales ante un nuevo mundo digital.* Mc Graw Hill : s.n., 1997.

Kendall, K. y Kendall, J. 1997. *Análisis y Diseño de Sistemas.* México : Prentice Hall, 1997. Tercera Edición.

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

Kruchten, Philippe. *A Rational Development Process.* s.l. : CrossTalk, 9 (7), STSC, Hill AFB, UT. págs. pp.11-16.

Larman, Craig. 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, 2005.

Marqués, José Manuel. 2005. Ingeniería del software. [En línea] 16 de 9 de 2005. [Citado el: 19 de 03 de 2008.] http://www.infor.uva.es/~jmmc/ingsoft/isprograma.html.

Martínez Juan, Francisco Javier. *Guías de Construcción de Software en Java con Patrones de Diseño.* Oviedo : Universidad de Oviedo.

Montesa Andrés, Jose Onofre. Tema 3. El Proceso de Desarrollo de Software. [En línea] Universidad Politécnica de Valencia. [Citado el: 24 de 3 de 2008.] www.upv.es/~jmontesa/eog/eog00-t3.ppt.

Oktaba, Hanna. Introducción a Patrones. *Facultad de Ciencias. UNAM.* [En línea] [Citado el: 19 de 02 de 2008.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html .

Övergard, Gunnar y Palmkvist, Karin. 2004. *Use Case Patterns and Blueprints.* s.l. : Addison Wesley Professional, 2004.

Palmero Sánchez, Maglema Ramona y Vázquez Baños, Yenier. 2007. *Sistema Generador de Mapas Temáticos y Gráficos Estadísticos.* La Habana : s.n., 2007.

Pazos Arias, Jose Juan. 2000. Ingeniería de Requisitos. [En línea] 16 de 2 de 2000. [Citado el: 2008 de 3 de 15.] http://www-gris.det.uvigo.es/~jose/doctorado/re/.

Pressman, Roger S. 1998. *Ingeniería de Software. Un enfoque práctico.* Cuarta Edición . Madrid : Mc Graw-Hill Interamericana de España S.A., 1998.

—. 2005. *Ingeniería del Software: Un enfoque práctico.* s.l. : Mc Graw Hill, 2005. Quinta Edición .

Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 2000. El Proceso Unificado de Desarrollo de Software. s.l. : Addison Wesley, 2000.

—. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana : Félix Varela, 2004. Vol. I.

—. 1998. *Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 1998.

—. 2004. *Proceso Unificado de Desarrollo de Software.* La Habana : Félix Varela, 2004. Vol. I.

—. 1999. *Unified Modeling Language—User's Guide.* s.l. : Addison-Wesley, 1999.

Sánchez Ríos, Sergio. 2007. Análisis y UML. Modelo conceptual. *Universidad Viña del Mar.* [En línea] Octubre de 2007. [Citado el: 27 de 03 de 2008.] http://www.uvmsf.cl/~ssanchez/images/Metodologias/Unidad4_MAD.pdf.

Santos, Ernesto. 1980. *Procesamiento de Datos.* Argentina : Procesamiento de Datos, 1980.

Schmuller, Joseph. 2000. *Aprendiendo UML en 24 horas.* México : Pearson Educación, 2000.

Tejera Hernández, Dayana Caridad y Sánchez Echevarría, Leidy Bárbara. 2007. *Ingeniería de Requisitos para el desarrollo del Sistema de Inventario Almacén (SIGIA) Módulo Nomencladores.* La Habana : s.n., 2007.

Zavala, R. 2000. *Diseño de un Sistema de Información Geográfica sobre Internet. Tesis de Maestría en Ciencias de la Computación.Universidad Autónoma Metropolitana-Azcapotzalco.* México : s.n., 2000.

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