# Universidad de las Ciencias Informáticas

Facultad 15


# Requerimientos y Diseño de una herramienta para la migración de los datos del proyecto Sistema Nacional Público para el Seguimiento de las Inversiones y Sectores (SINAPSIS).

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

**Autores:** Armando Labrada Leyva

Jorge Bárbaro Rodríguez Pérez

**Tutor:** José Miguel Cazorla Santana

**Ciudad de la Habana**

**Junio 2010**

**DECLARACIÓN DE AUTORÍA**

Declaramos ser los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Armando Labrada Leyva

_____

(Autor)

Jorge Bárbaro Rodríguez Pérez

_____

(Autor)

José Miguel Cazorla Santana

_____

(Tutor)

**Agradecimientos**

*A mi país y a la revolución por darme la posibilidad de graduarme como ingeniero. A mis padres queridos que siempre me han apoyado en todo lo que he necesitado. A mi querido hermano por brindarme sus buenos consejos a cada momento. A mi tutor que me ha aclarado cuanta duda me ha surgido, gracias por ser mi guía. En fin a todos mis amigos que de una forma u otra contribuyeron en que yo alcanzara este logro. A todos ellos muchas gracias, gracias por los momentos que pasamos juntos ya sea estudiando o divirtiéndonos. En general a toda mi familia y mis amistades, muchas gracias.*

Armando Labrada Leyva

*A mis padres y mi familia por contribuir en mi formación, por la confianza y el cariño que siempre me han dado.*

*A nuestro tutor por su apoyo y ayuda constante, y por trasmitirnos nuevos conocimientos para la vida futura como profesional.*

*A todos los profesores y personas que han influido en mi educación durante toda mi carrera como estudiante.*

*A los profesores, compañeros y amigos que ayudaron de una u otra forma al desarrollo de este trabajo.*

*A todos muchas gracias…*

Jorge Bárbaro Rodríguez Pérez

**Dedicatoria**

*Sin duda ninguna a las personas más importantes para mí en la vida. A mis padres Armando y Lorenza por hacer de mí el hombre que soy hoy, por brindarme su cariño, su amor y dedicación en toda mi vida, por ser un ejemplo a seguir. A mis primos Amaury y Marveidis que son como mis hermanos que siempre me han dejado la guía por donde yo debía coger, gracias por permitirme ser su hermano, por brindarme sus consejos y servirme como un ejemplo de bien. A mis abuelos por darme la posibilidad de verme convertido en ingeniero. A mi tía Norma y su esposo Rafael por brindarme amor en su casa. En fin a toda mi familia. A todos mis amigos que han sido siempre mi mejor compañía.*

Armando Labrada Leyva

*A mi madre Isabel, mi padre Jorge, mi hermano Josiel, mi novia Adri y toda mi familia por estar siempre a mi lado y confiar en mí, por el amor que siempre me han dado.*

*A mis amigos del barrio y los de la escuela por compartir momentos felices de mi vida.*

*A todos muchas gracias por ayudarme a realizar este gran sueño de mi vida.*

Jorge Bárbaro Rodríguez Pérez

## RESUMEN

La presente tesis para optar por el título de “Ingeniero en Ciencias Informáticas”, titulada “Requerimientos y Diseño del módulo Migración de Datos del proyecto Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS)”, fue realizada a través del perfil del rol de analista de un proyecto de software.

Debido a la cooperación existente entre los países de Venezuela y Cuba a través del Convenio Integral de Cooperación, se han hecho diferentes proyectos productivos con especialistas de ambos países, dentro de este convenio se encuentra el proyecto Sistema Nacional Público para el Seguimiento de las Inversiones y Sectores (SINAPSIS), debido a la deficiencia existentes en el área de la planificación del sector público venezolano así como el ineficiente manejo de los recursos asignados por el estado, se decide desarrollar este proyecto para tener un mejor control y seguimiento. El proyecto cuenta con varios módulos, y dentro de ellos se encuentra el de Migración de Datos, el cual necesita una herramienta para la migración de los datos.

Con el desarrollo de este trabajo se obtienen los requerimientos y el diseño de dicho módulo, también se hace un estudio de las diferentes herramientas que se utilizan para la migración de datos, metodologías de desarrollo de software, herramientas de modelado, lenguajes de programación, sistemas gestores de base de datos, Requerimientos y Diseño definidos en la metodología que se utilizó para realizar el trabajo, también se aplicó el proceso de validación que ofrece Calidad UCI, a través de listas de chequeo, para obtener un producto con calidad.

## Índice de Figuras

## Contenido

## INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) son incuestionables, y forman parte de la cultura tecnológica existente y con la que se debe convivir. Amplían las capacidades físicas, mentales y las posibilidades de desarrollo social. Las TIC's no solamente es la informática y sus tecnologías asociadas, telemática y multimedia, sino también los medios de comunicación de todo tipo: los medios de comunicación social e interpersonal. Las TIC's contribuyen a la rápida obsolescencia de los conocimientos y a la emergencia de nuevos valores, provocando continuas transformaciones en las estructuras económicas, sociales y culturales, e incidiendo en casi todos los aspectos de de la vida.

La aceleración de las economías mundiales en el tema del desarrollo del software se hacen cada vez más competitivas, y las cantidades de software desarrollados es cada vez mayor. Estos software necesitan en gran medida mantener la integridad de los datos existentes a la hora de sustituir las bases de datos y ello conlleva a la migración de los mismos.

Por lo que se habla de migración de datos cuando se refiere al traspaso de información entre bases de datos. Al actualizar una nueva versión de una base de datos o de una aplicación, o al cambiar a un nuevo sistema, los datos necesitan ser preservados en este nuevo sistema. El propósito de la migración de datos es transferir datos existentes al nuevo ambiente. Los datos necesitan ser transformado a un formato conveniente para el nuevo sistema, mientras que se preserva la información presente en el viejo.

La migración de datos de aplicaciones y bases de datos antiguas a nuevas es un subproducto necesario del crecimiento organizacional y de la adaptación a las cambiantes necesidades de negocios. Pero el mezclar datos con diferentes formatos, estructuras y parámetros continúa siendo una de las prácticas más riesgosas de las áreas de Tecnología de Información, resultando frecuentemente en grandes retrasos, costos elevados más de lo pronosticado e inclusive en fallas completas del proyecto. Si la migración involucra una aplicación o iniciativa de misión crítica, el impacto puede ser desastroso.

En la actualidad, numerosas herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador), son capaces de generar todo o parte del código que implementa una aplicación a partir de la información que se introduce en los modelos del método que se esté utilizando. Así por ejemplo, herramientas como Rational Rose, Together o System Architect, son capaces de generar esqueletos de programas en lenguajes de programación Java, C++, VBasic y scripts en SQL para generar las bases de datos necesarias a partir de la información estructural que se introduce mediante dichas herramientas en los diagramas de clases.

Otras herramientas CASE como OO-Method/CASE u Oblog/CASE, son capaces de generar aplicaciones completas a partir de la información de modelado que se introduce en sus modelos gracias a su sólido soporte formal. Todas las herramientas CASE anteriores no ofrecen un buen soporte al paso del tiempo. La introducción de un nuevo requerimiento en las aplicaciones en funcionamiento, siguiendo algún método de gestión de cambios, consiste en introducirlo en los modelos pertinentes para posteriormente y aprovechando las capacidades generadoras de las herramientas CASE, regenerar la aplicación y el esquema de la base de datos acorde con el modelo del sistema actualizado. Finalmente, existen dos esquemas conceptuales del sistema de información, uno inicial con la definición del sistema antes de introducir los nuevos requerimientos y uno final en el que ya se han introducido. Además, existen dos bases de datos, una inicial con toda la información que se ha generado mientras la aplicación ha estado en funcionamiento, y una final que satisface los nuevos requerimientos del sistema, sin información.

En la actualidad existen herramientas informáticas que permiten realizar la migración de datos, pero todas ellas tienen limitaciones y restricciones que las hacen parcialmente ineficientes.

El sistema Nueva Etapa no fue desarrollado utilizando una metodología de desarrollo de software y no cuenta con la documentación necesaria que facilite su comprensión. Esto dificulta su optimización y mantenimiento. Para darle solución a la problemática planteada anteriormente se decidió desarrollar el Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS), un sistema informático que sustituirá completamente el anterior y se dividirá en siete módulos, entre los que se encuentra el de Migración de Datos.

Después de tener identificados y documentados los procesos de negocios del módulo se deben elaborar un conjunto de artefactos (diagramas, modelos) que le proporcionen a los desarrolladores las bases para implementar el módulo exitosamente.

**Problema Científico:**

Los artefactos actuales del módulo Migración de Datos del sistema SINAPSIS no garantizan el comienzo y continuidad de la implementación del módulo.

**Objeto de Estudio:**

El Proceso de Desarrollo de Software.

**Campo de Acción:**

Requerimientos y Diseño.

**Hipótesis:**

Si se logra transformar las necesidades del cliente en un lenguaje entendible por los desarrolladores, se podrá realizar la implementación de la herramienta, y de esa forma se logrará realizar en el proyecto SINAPSIS la migración de los datos.

**Objetivo General:**

Generar los artefactos necesarios del módulo Migración de Datos del sistema SINAPSIS que permitan el comienzo y continuidad de la implementación del módulo.

**Tareas:**

1: Estudio de las diferentes herramientas que se utilicen para la migración de datos.

2: Estudio de las diferentes metodologías que existen y proponer la más adecuada para el análisis y diseño de la herramienta.

3: Estudio de las herramientas de desarrollo para obtener los requerimientos y el diseño.

4: Identificación de los requerimientos funcionales y no funcionales del sistema.

5: Elaboración de la Especificación de Requerimientos.

6: Confección de los diagramas de clases del diseño y los diagramas de secuencia correspondientes.

7: Validación de los resultados obtenidos a través de técnicas de validación.

**Métodos**

**Métodos teóricos**

**Histórico-lógico:** Este método se utilizó para determinar las tendencias presentes en el diseño del software en el mundo, así como para analizar experiencias anteriores en el desarrollo de herramientas para la migración de datos.

**Analítico-sintético:** Este método se utilizó para estudiar toda la bibliografía seleccionada sobre el tema de investigación, procesarla y arribar a conclusiones, permitiendo así hallar las características principales de los patrones de diseño posibles a utilizar.

**Inductivo-deductivo:** Este método permitió que a partir del estudio de las distintas tecnologías y metodologías posibles a utilizar para los requerimientos y el diseño, se arribará a proposiciones de estilos y patrones de diseño específicos.

**Métodos empíricos**

**Observación**: Se pudo obtener la información directa del comportamiento del fenómeno que está siendo investigado tal y como actúa en la realidad los diferentes procesos de la migración de datos.

**Entrevista:** Este método permitió obtener experiencias, criterios y conocimientos sobre las tendencias y patrones de diseños más importantes usados en la universidad según los especialistas consultados.

Para lograr la comprensión y claridad de los contenidos de la investigación realizada se ha estructurado el documento en tres capítulos.

**En el Capítulo 1:**"*Fundamentación teórica"*, podrá encontrar una serie de aspectos relacionados con la necesidad de una herramienta para la migración de datos del proyecto SINAPSIS. Además se estudian las herramientas, metodologías y tecnologías a utilizar.

**En el Capítulo 2:**"*Requerimientos y diseño*", se muestra la solución escogida para obtener los requerimientos y el diseño del módulo de migración de datos del proyecto SINAPSIS.

**En el Capítulo 3:**"*Validación de la solución propuesta"*, se aplican una serie de métricas de diseño y de requerimientos, y se analizan los resultados para determinar la calidad del modelo del diseño y del modelo del sistema realizado para el Subsistema de migración de datos del proyecto SINAPSIS.

## Capítulo 1

### 1.1 Introducción

En este capítulo se pretende abordar los aspectos y conceptos relacionados con la herramienta a utilizar para la migración de los datos. Se hace un análisis del comportamiento de las funcionalidades a realizar en el módulo. Se tratan aspectos concernientes a la Ingeniería de Requerimientos y de las métricas que aseguran la factibilidad y la calidad de la especificación de requerimientos y el modelo de diseño. Se realiza un estudio de las metodologías de desarrollo de software existentes y se explican las herramientas seleccionadas para dar solución a la problemática propuesta, pretendiendo dejar sentadas las bases teóricas para obtener los requerimientos y diseño del software.

### 1.2 Base de datos

El término de Base de Datos fue escuchado por primera vez en 1963, en un simposio celebrado en California EUA. Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más **columnas** y **filas**. Las columnas guardan una parte de la información sobre cada elemento que se quieran guardar en la tabla, cada fila de la tabla conforma un registro. (1)

Una Base de Datos o Banco de Datos se puede definir como un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. La utilización de bases de datos como plataforma para el desarrollo de Sistemas de Aplicación en las Organizaciones se ha incrementado notablemente en los últimos años, se debe a las ventajas que ofrece su utilización, algunas de las cuales se comentan a continuación:

- Globalización de la información: permite a los diferentes usuarios considerar la información como un recurso corporativo que carece de dueños específicos.
- Eliminación de información inconsistente: Si el sistema está desarrollado a través de archivos convencionales, una cancelación de compra por ejemplo deberá operarse tanto en el archivo de facturas del Sistema de Control de Cobranza como en el archivo de facturas del Sistema de Comisiones.
- Permite mantener la integridad de la información como cualidad altamente deseable y tiene por objetivo almacenar correctamente la información.
- Independencia de datos como factor esencial en la rápida proliferación del desarrollo de Sistemas de Bases de Datos. La independencia de los datos implica un divorcio entre programas y datos.

## 1.3 Sistema Gestor de Base de Datos (SGBD).

Los Sistemas Gestores de Base de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. En los textos que tratan este tema, o temas relacionados, se mencionan los términos Sistema Gestor de Base de Datos (SGBD) y Sistema Manejador de Base de Datos (DBMS).(2)

El propósito general de los Sistemas de Gestión de Base de Datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información. Estos sistemas de Gestión de Base de Datos están compuestos de un lenguaje de definición de datos (DDL: Data Definition Language), de un lenguaje de manipulación de datos (DML: Data Manipulation Language) y de un lenguaje de consulta (SQL: Structured Query Language).

Este tipo de software tienen características comunes que los definen tales como:

**Índices**

El empleo adecuado de índices en una relación acelera el acceso a la información, pero consume espacio considerable, es por esto que vale la pena hacer un análisis cuidadoso de cuáles atributos requieren ser indexados.

**Niveles**

A continuación se explica el funcionamiento de cada uno de los niveles:

**Interno:** cómo se almacenan y recuperan los datos (único).

**Externo:** cómo perciben los datos, los usuarios (muchos).

**Conceptual:** enlace entre los anteriores.

### 1.3.1 PostgreSQL 8.4

PostgreSQL es un sistema gestor de base de datos objeto relacional libre, liberado bajo la licencia BSD. Como muchos otros proyectos Open Source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo, dicha comunidad es denominada el PostgreSQL Grupo Global de Desarrollo (PGDG), sus siglas en inglés se definen como: PostgreSQL Global Development Group.

PostgreSQL ha tenido una larga evolución, comenzando con el proyecto Ingres en la Universidad de Berkeley. (3)

Este proyecto, liderado por Michael Stonebraker, fue uno de los primeros intentos en implementar un motor de base de datos relacional.

### 1.3.2 Oracle Instant Client

Oracle es básicamente una herramienta Cliente-servidor para la gestión de bases de datos, es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que solo se vea en empresas muy grandes y multinacionales, por norma general. Es un sistema manejador de Bases de Datos Relacional que hace uso de los recursos de los sistemas informáticos en todas las arquitecturas de hardware, lo que permite garantizar su aprovechamiento en ambientes cargados de información, por su capacidad de almacenar y acudir a los datos de forma recurrente.

La última versión es la 11g, pero la aplicación se desarrolla para la versión 10g. Oracle Instant Client es la librería que permite el establecimiento de las conexiones con Oracle, el cual tiene varias opciones en dependencia de la versión de Oracle que se esté usando (8,9i, 10g y 11g) y de la arquitectura. (Ya sea de 32 o 64 bits). (4)

## 1.4 Tendencias actuales de las aplicaciones de migración de Oracle a PostgreSQL.

Las herramientas existentes en el mundo se pueden clasificar en:

**Propietarias – Comerciales:**

- EnterpriseDB con su PostgreSQL Plus (USA).
- CommandPrompt, Inc. con su Mammoth Postgresql (USA).
- Fujitsu con sus servicios de consulta, entrenamiento y migración (Japón).
- Hub.org con sus servicios de alojamiento Web (Canadá).

**Libres:**

- Ora2Pg
- OraLink

### 1.4.1 Herramientas libres de migración de bases de datos Oracle a PostgreSQL.

- **OraLink**

OraLink es una pasarela para acceder a Oracle directamente de bases de datos PostgreSQL. Puede ser utilizado para la integración, la replicación, o simplemente para acceder a Oracle utilizando datos PostgreSQL. Esta herramienta está escrita en C, se acerca un poco más a lo que necesita pero debido que se ha descontinuado su desarrollo no resulta factible el uso de la misma.

- **Ora2Pg**

Ora2Pg es un módulo de Perl para exportar una base de datos Oracle a un esquema de PostgreSQL. Esta aplicación se conecta a su base de datos Oracle, extrae su estructura, y genera un script SQL que puede cargar en su base de datos PostgreSQL.

La misma hace un backup de todo el esquema (tablas, vistas, secuencias, índices, privilegios) con todas las llaves ya sean primarias, foráneas y únicas exportadas a código PostgreSQL sin necesidad de editar el SQL generado. Asimismo, se puede efectuar el volcado de datos Oracle en la base de datos PostgreSQL como proceso en línea o en un archivo. Puede elegir qué columnas se pueden exportar para cada tabla. Esta herramienta es la que más promete a la hora de hacer efectiva la migración. Presenta disímiles ventajas con respecto a las anteriormente expuestas como:

- La división de columnas para cada tabla a exportar a PostgreSQL.
- Trabaja con versiones distintas de Oracle (8,9i y 10g).
- Posee una extensa guía de migración muy intuitiva en su sitio oficial.

Pero Ora2Pg presenta varias desventajas por las cuales surge la necesidad de desarrollar una nueva herramienta libre como son:

- Al ser desarrollado en Perl, se está atado al largo ciclo de desarrollo propio del lenguaje a expensas de una nueva versión del mismo; además de que se hace un poco difícil también incorporarle una interfaz gráfica para una mejor interacción con el usuario.
- Existe muy poca documentación de la aplicación, y la disponible y considerada la más actualizada está en el idioma francés.
- La aplicación no está disponible en la mayoría de distribuciones de GNU/Linux, sólo existen paquetes para Debian GNU/Linux, Ubuntu, Red Hat y Fedora.

- Los algoritmos de conversión con que cuenta pueden mejorarse a la hora de realizar la conversión a código SQL compatible con PostgreSQL aprovechando al máximo las nuevas características del gestor como el mejor soporte para las vistas materializadas, particionamiento de tablas, espacios de tablas, entre otras.

Por todo lo anteriormente expuesto, se prosigue a desarrollar una nueva herramienta, la cual será escrita en java, que tenga como base a la aplicación Ora2Pg por las excelentes ideas que promueve y por la experiencia de sus desarrolladores. La presente herramienta para la migración de los datos en el sistema SINAPSIS trata de darle solución a la totalidad o a la mayoría de los problemas con que cuenta Ora2Pg.

### 1.5 Evolución del análisis y diseño del software. Estado Actual.

El término de análisis de software en su principio con un enfoque estructurado surge como complemento a otra problemática, el diseño estructurado. El término de análisis estructurado fue ideado por Douglas Ross. En la década de los 80, Ward y Mellor y más tarde Hatley y Pirbhai introdujeron las aplicaciones en tiempo real, con las cuales se consiguió un método de análisis más robusto para resolver problemas informáticos. A finales de la década de los 80 con el auge de los mecanismos orientado a objetos, surgen un gran número de técnicas de modelado de análisis y diseño, pero esta vez con este nuevo enfoque. No es hasta que Grady Booch, James Rumbaugh e Ivar Jacobson, comenzaron a colaborar en fusionar las mejores técnicas de cada uno de sus métodos de análisis y diseño orientados a objetos, dando paso a lo que se conoce hoy en día como Lenguaje Unificado de Modelado (UML) del inglés Unified Modeling Language, el cual se ha convertido en el método más usado en los últimos años (5). En la actualidad gracias al uso del UML junto a herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) de desarrollo que facilita el modelado del análisis y diseño en el desarrollo de software aplicando metodologías de desarrollo de software como es el caso de RUP, XP u otras de las existentes, se ha logrado un desarrollo de las técnicas de análisis y diseño y con ello un mayor perfeccionamiento en el proceso de desarrollo de software.

### 1.6 Metodologías de Desarrollo de Software.

El desarrollo de un software debe regirse por una guía que indique en cada momento de su ciclo de vida quien debe hacer qué, cómo y cuándo. El análisis y diseño del software en construcción es de suma importancia y es desarrollado con diferentes enfoques según la metodología de desarrollo de software usada. En este epígrafe se analizarán los principales elementos que exponen en el tema del análisis y

diseño, algunas de las metodologías de desarrollo de software más usadas del mundo, como son: Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF). (5)

- **Extreme Programming (XP)**

Programación extrema del inglés Extreme Programming (XP) es una de las conocidas metodologías de desarrollo de software ágiles. El desarrollo con XP se efectúa de forma incremental, sus principios fundamentales se pueden resumir en los siguientes (13):

- **Comunicación:** Comunicación total a todos los niveles de trabajo. Se trabaja en grupos de dos personas por ordenador pero con total comunicación en todos los momentos entre todos los grupos.
- **Usuario:** El usuario siempre está en mente. Se han de satisfacer sus necesidades pero nada más.
- **Simplicidad:** Lo más simple es lo mejor, funciona mejor, más rápido, es más adaptable, más barato y además es más entendible.
- **YAGNI:** "You aren't gonna need it" (No lo vas a necesitar). No hagas nada que creas que en el futuro puede ser útil porque probablemente no lo vas a necesitar. Es una pérdida de tiempo.
- **OAOO:** "Once and only once" (Una única vez). Las cosas se hacen una sola vez.

Estos principios se ven complementados en lo que llaman "sus 12 buenas prácticas": Planificación, Versiones Pequeñas, Sistema Metafórico. (Metaphor), Diseños simples, Testeos Continuos, Refactoring, Programación en parejas, Propiedad colectiva del código, Integración continua, 40 horas por semana, El cliente en su sitio, Estándares de codificación.

El análisis se efectúa de forma muy ligera, mientras que el diseño (Figura 1), si se lleva a profundidad como en la mayoría de las metodologías. El análisis se transforma en la exposición por parte del cliente de las "user-stories", para lograr un entendimiento de lo que se quiere. Luego continua el diseño global del sistema, en el que se profundiza para que los desarrolladores sepan exactamente que van hacer.

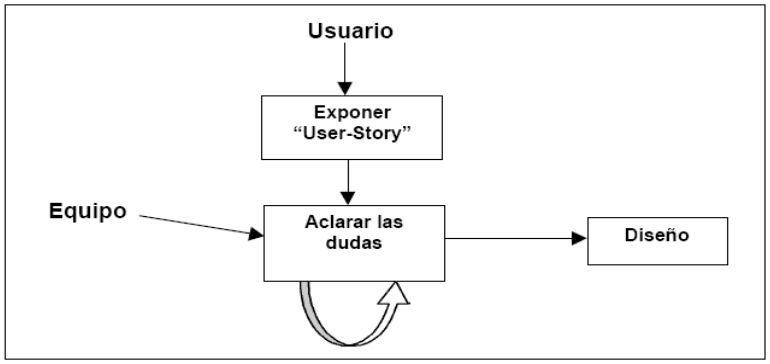



**Figura 1. Actividades del diseño en XP.**

- **Microsoft Solution Framework (MSF)**

MSF es una metodología flexible. Se centra en el modelo de cascada, usando puntos de control para el paso entre sus fases y en el modelo espiral para refinar continuamente los requerimientos y las estimaciones del proyecto. Se compone de varios modelos, encargados de planificar las partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el Modelo de Aplicación. El proceso de desarrollo en MSF consta de fases: Visión, Planeación, Desarrollo, Estabilización e implantación.


**Figura 2. Fases de MSF.**

MSF enfoca el análisis y diseño a través del Modelo de Diseño del Proceso en la fase de planeación. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los usuarios, el equipo y los desarrolladores.

- **Rational Unified Process (RUP)**

RUP es una metodología de desarrollo de software que da la guía para transformar los requerimientos que se obtienen una vez definida la necesidad de automatización y mejoras que requiere y necesita el cliente, en casos de uso que posteriormente se convertirán en conjunto de funcionalidades que le dan estructura al sistema desarrollado y que trae resultados visibles al usuario o cliente. Está definido en función de flujos de trabajos y fases (Figura 3), las cuales llevan el software desde su concepción hasta su entrega al usuario final.

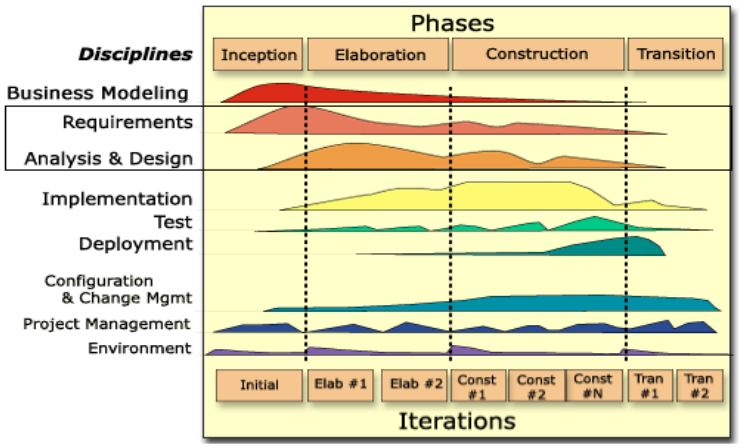



**Figura 3. Proceso unificado de desarrollo de software (fases y flujos de trabajos).**

Para RUP el análisis y diseño se concreta a través de sus flujos de trabajos. Parte desde el modelado del negocio, donde se modelan los procesos del negocio, luego comienza la etapa de requerimientos, donde se identifican las funcionalidades que se automatizarán del negocio definido, luego inicia el análisis y diseño el cual tiene como entrada el modelo del sistema definido en el flujo de trabajo anterior. Define que en primer lugar se debe determinar una arquitectura candidata, luego a partir del modelo del sistema hacer el análisis de los casos de uso determinados y por último se deben diseñar los componentes pertinentes. (14)

Luego de un estudio realizado a tres metodologías de desarrollo (RUP, XP y MSF) se decidió utilizar RUP. A continuación se exponen las razones por las que fue escogida esta metodología.

El proceso unificado conocido como RUP, es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación.

RUP identifica claramente a los profesionales (actores) involucrados en el desarrollo del software y sus responsabilidades en cada una de las actividades. Además, explícitamente indica qué actor es responsable de qué artefacto en cada actividad.

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

## 1.7 Herramientas Case.

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador), son aplicaciones que facilitan el desarrollo de software, reduciendo el esfuerzo, el costo y el tiempo, además de estructurar la documentación asociada a los artefactos generados. A continuación se exponen las principales características de tres de las herramientas CASE más utilizadas en el mundo en la actualidad. (15)

### 1.7.1 Enterprise Architect.

Es una herramienta CASE orientada a objetos que provee su alcance para el desarrollo de sistemas, administración de proyectos y análisis de negocios. Maneja totalmente el ciclo de vida de desarrollo de software, utilizando el UML como leguaje de modelado. Facilita y soporta el levantamiento de requerimientos, el análisis y diseño, las pruebas y mantenimiento del software en desarrollo. Soporta diferentes lenguajes de desarrollo, incluyendo ActionScript, C, C++, C# y VB.NET, Java, Visual Basic 6, Python, PHP, XSD, WSDL y otros más. Gestiona la ingeniería de código, normal e inversa, además de una efectiva documentación compatible con Microsoft Word. Dentro de las funcionalidades que soporta el Enterprise Architect están: diagramas UML, CU, modelos lógico, dinámico y físico, extensiones personalizadas para modelado de procesos, documentación de alta calidad compatible con MS Word, modelado de datos, Ingeniería directa de base de datos a DDL e ingeniería inversa de base de datos desde ODBC, Soporte de pruebas y multi-usuario con sistema de seguridad.

### 1.7.2 Rational Rose.

Es una herramienta case que soporta el flujo completo de desarrollo de software. Está basada en la metodología de desarrollo RUP. Da soporte al UML. Es un entorno de modelado que permite generar código a partir de modelos ADA, ANSI C++, C++, CORBA, Java/J2EE, Visual C++ y Visual Basic. Facilita

la documentación de los artefactos generados en el proceso de desarrollo de software. Estructura el proyecto en función de las vistas de la arquitectura que plantea la metodología de desarrollo RUP. (16)

Dentro de las funcionalidades que soporta el Rational Rose están: diagramas UML, desarrollo orientado al modelado, la generación de código ADA, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables, capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación, publicación web y generación de informes para optimizar la comunicación dentro del equipo.

### 1.7.3 Visual Paradigm.

Es una herramienta CASE que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo de desarrollo de software, ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. (17)

Dentro de las funcionalidades que soporta el Visual Paradigm están: Administración de Requerimientos, modelado de procesos del negocio, modelado de base de datos, generación de código e Ingeniería inversa.

Visual Paradigm posibilita una mejor integración entre todos los involucrados en el desarrollo del software, brindándole la posibilidad de organizar los diagramas y documentación asociada al desarrollo del proyecto.

A continuación aparecen razones por las cuales una organización debe elegir VP-UML y SDE:

1. Permite incorporar dibujos de Visio en cualquier diagrama UML: A diferencia de otras herramientas CASE-UML, VP-UML y SDE prolongan esta limitación permitiendo incorporar dibujos de Visio en cualquier diagrama de UML, pudiendo modelar un dominio específico de hardware, software, redes, componentes, etc.
2. Integración perfecta con los principales IDEs: SDE está disponible en varios IDEs incluyendo Microsoft Visual Studio, Borland JBuilder, Eclipse/IBM WSAD, NetBeans/Sun ONE, IntelliJ IDEA y JDeveloper, lo que puede proporcionar un entorno que integra todo el modelo de código desplegando el proceso de desarrollo de software.
3. Soporta múltiples plataformas: No importa el Sistema Operativo que esté en uso, VP-UML y SDE se puede ejecutar en equipo y están disponibles en muchas plataformas como Windows, Linux, Mac OS X y Java Desktop.

4. El VP-UML es una herramienta de modelado multiplataforma que ofrece una interfaz amigable, permite la integración con varios IDE de desarrollo, permite Ingeniería Inversa lo que posibilita entregar al cliente un producto mejor documentado, y por último incluye un panel para el diseño de la interfaz de usuario por lo que no es necesaria la integración con Visio.

## 1.8 Lenguaje de Modelado.

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un diseño de software orientado a objetos. Algunas organizaciones los usan extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. Algunos metodólogos del software orientado a objetos distinguen tres grandes "generaciones" cronológicas de técnicas de modelado de objetos (18):

- En la primera generación, se incluye a autores y técnicas como Rumbaugh, Jacobson, Booch, los métodos formales, Shlaer-Mellor y Yourdon-Coad.
- En la segunda generación se realizaron múltiples intentos para integrar dichas técnicas en marcos coherentes tales como FUSIÓN. Se empezaba a reconocer los beneficios que la estandarización de las técnicas conllevaría: hacer las cosas de una manera adecuada, que permitiría un lenguaje y unas prácticas comunes entre los diferentes desarrolladores.
- La tercera generación consiste en intentos creíbles de crear dicho lenguaje unificado por la industria, cuyo mejor ejemplo es UML.

### 1.8.1 Lenguaje Unificado de Modelación (UML)

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema, incluye aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos. (18)

De forma general las principales características son:

1. Lenguaje unificado para la modelación de sistemas.
2. Tecnología orientada a objetos.

3. El cliente participa en todas las etapas del proyecto.
4. Corrección de errores viables en todas las etapas.
5. Aplicable para tratar asuntos de escala inherentes a sistemas complejos de misión crítica, tiempo real y cliente/servidor.

Se utiliza el lenguaje UML por que indica qué es lo que supuestamente hará el sistema, no precisamente cómo lo hará. Pero constituye una guía y una representación de las especificaciones del sistema, ayudando a tomar decisiones para lograr un sistema que cumpla con todas las expectativas del cliente y los usuarios.

## 1.9 Lenguaje Programación.

Un lenguaje de programación es usado para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Para desarrollar la herramienta se ha realizado un pequeño estudio sobre diferentes tipos de lenguajes de programación como son: Java, C++, C# y SQL.

### 1.9.1 Lenguaje C++.

Con el lenguaje C++ se buscó incluir completamente al lenguaje C pero al mismo tiempo se le agregó algunas características como: programación orientada a objetos, excepciones, sobrecarga de operadores, templates o plantillas, para así mejorar dicho lenguaje y hacer más fácil el trabajo para los programadores.

### 1.9.2 Lenguaje Java.

El lenguaje Java heredó características de C y C++, explícitamente eliminando aquellas que para muchos programadores (según los diseñadores) resultan excesivamente complejas e inseguras, y así facilitar el trabajo de los programadores. En la actualidad su uso es promovido para el desarrollo de aplicaciones empresariales del lado del servidor, especialmente a través del estándar J2EE, así como en dispositivos móviles (a través del estándar J2ME.)

### 1.9.3 Lenguaje C#.

El lenguaje C# es una versión avanzada de C y C++ y se ha diseñado especialmente para el entorno .NET. Es un nuevo lenguaje orientado a objetos empleado por programadores de todo el mundo para desarrollar aplicaciones que se ejecuten en la plataforma .NET. Con el C# se puede desarrollar todo tipo

de proyectos de aplicaciones cliente/servidor. El C# mezcla la potencia de C, las capacidades de orientación a objetos de C++ y la interfaz gráfica de Visual Basic.

## 1.10 Herramientas de Desarrollo.

### 1.10.1 NetBeans

NetBeans es una herramienta de código abierto, fundado por Sun MicroSystems en 2000. NetBeans es una utilidad (de código abierto) con la que se podrá compilar, depurar y ejecutar los programas realizados. Esta herramienta está enfocada para el desarrollo en el lenguaje Java, pero es válido para otro lenguaje de programación. Es uno de los mejores IDE para programar en Java. Bastante sencillo, cómodo y muy configurable. Sitio oficial de NetBeans: http://www.netbeans.org.

### 1.10.2 Eclipse

El IDE de desarrollo Eclipse está considerado como uno de los mejores entornos de programación, desarrollado por la Fundación Eclipse, una organización sin ánimos de lucro que publica oficialmente todas las versiones nuevas sobre esta herramienta, su objetivo principal es el desarrollo de una plataforma libre Integrated Development Environment en inglés (Entorno Integrado de Desarrollo) de desarrollo que contenga todas las herramientas necesarios para desarrollar el ciclo completo de un software determinado.

## 1.11 Ingeniería de Requerimientos

La Ingeniería de Requerimientos (IR), como disciplina iniciadora del proceso de desarrollo de software, tiene como principal objetivo garantizar la especificación de un sistema que cumpla con las expectativas y necesidades del cliente. En esta se identifican dos aspectos fundamentales, el primero, cuál es el propósito del sistema que se va a desarrollar y el segundo, el contexto en el que será usado. (5)

### 1.11.1. Actividades de la Ingeniería de Requerimientos.

El proceso de IR puede ser descrito a través de 4 actividades las cuales son:

- **Identificación de Requerimientos**

Esta actividad se refiere a la captura y descubrimiento de los requerimientos que deberán ser implementados. Es una actividad más “humana” que técnica, se identifican a los interesados

(stakeholders) y se establecen las primeras relaciones entre ellos y el equipo de trabajo. En principio parece una tarea relativamente fácil, preguntar al cliente, usuarios y a todos aquellos involucrados en el negocio y sean expertos en el tema, qué es lo que desean que haga el sistema que se va a desarrollar. (5)

- **Análisis de Requerimientos y Negociación**

Es la actividad de la IR en la cual se estudia la información extraída durante la actividad anterior, para identificar la presencia de áreas no detectadas, requerimientos contradictorios y peticiones que aparecen como vagas e irrelevantes. Una vez recopilados los requerimientos, el producto obtenido configura la base del análisis de requerimientos, estos se agrupan por categorías y se organizan en subconjuntos, se estudia cada uno en relación con el resto, se examinan si existen conflictos en cuanto a su consistencia, completitud y ambigüedad y se clasifican en base a las necesidades de los clientes/usuarios. (5)

- **Especificación de Requerimientos**

La especificación de requerimientos, puede ser un documento escrito, un modelo gráfico, una colección de escenarios de uso, un prototipo o una combinación de lo anteriormente citado, donde se describe lo qué hay que desarrollar, no el cómo ni el cuándo. (5) El objetivo de esta actividad es obtener un documento de especificación de requerimientos de software (ERS) que defina, de forma completa, precisa y verificable, los requerimientos que debe cumplir el sistema, tanto funcional como no funcional. No debe incluir requerimientos innecesarios, que no hayan sido solicitados por el cliente, ni incluir detalles sobre el diseño del sistema.

- **Validación de Requerimientos**

El resultado del trabajo realizado es consecuencia de la Ingeniería de Requerimientos y es evaluada su calidad en la fase de validación. La validación de requerimientos examina las especificaciones para asegurar que todos los requerimientos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. (5)

### 1.11.2. Técnicas para el levantamiento de requerimientos.

La Ingeniería de Requerimientos tiene un valor elevado en el desarrollo de software, por lo tanto es necesario realizar la misma de la forma correcta. Es por esta razón que a continuación se fundamentan las técnicas más utilizadas para llevar a cabo las actividades de la IR, las cuales han sido aplicadas al presente trabajo. (8)

- **Entrevistas**

Es una de las técnicas de identificación más usada, la cual consiste en establecer un canal de comunicación directa entre las personas destinatarias del sistema y el Equipo de Desarrollo. Las entrevistas planeadas generalmente se dan de forma interactiva y realimentada. Las entrevistas son dirigidas normalmente por el personal más experto del Equipo de Desarrollo, quienes junto con un equipo interdisciplinario de profesionales de otras áreas, como la psicología y el derecho, son los encargados de orientar las entrevistas de tal forma que la información obtenida a través de ellas sea relevante al proceso. En esta técnica se pueden identificar tres fases: la preparación, la realización y el análisis de la información obtenida.

- **Sistemas Existentes**

Esta técnica consiste en analizar distintos sistemas ya desarrollados que estén relacionados con el sistema a ser construido. Algunas de las ventajas de esta técnica son:
Se pueden analizar las interfaces de usuario, observando el tipo de información que se maneja y cómo es manejada. Y es útil analizar las distintas salidas que los sistemas producen (listados y consultas).

Cuando se utiliza esta técnica se puede realizar a priori sin que intervenga el cliente/usuario para ello, existen en Internet cantidad de demos de productos que pueden resultar similares, también se pueden establecer contactos con profesionales que desarrollan sistemas de características comparables, aunque esto requiere de cierto grado de trabajo (investigación y análisis).

- **Tormenta de ideas (Brainstorming)**

Esta es una técnica que se usa para generar ideas. La intención en su aplicación es la de generar la máxima cantidad posible de requerimientos para el sistema. No hay que detenerse en pensar si la idea es o no del todo utilizable.

A veces ocurre que una idea resulta en otra idea, y otras veces se pueden relacionar varias ideas para generar una nueva.

- **Arqueología de documentos**

Con la aplicación de esta técnica se tratan de determinar posibles requerimientos sobre la base de inspeccionar la documentación utilizada por la empresa; por ejemplo, manuales de procedimientos, reglamentos, boletas y facturas. Esta técnica sirve más que nada como complemento de las demás técnicas y ayuda a obtener información que de otra manera sería sumamente difícil conseguir.

- **Prototipos**

Los prototipos surgen para validar los requerimientos hallados. Estos son simulaciones del posible producto, que luego son utilizados por el usuario final, lo que permite conseguir una importante retroalimentación en cuanto a si el sistema diseñado en base a los requerimientos recolectados, le permite al usuario realizar su trabajo de manera eficiente y efectiva. Los prototipos se pueden clasificar en:

Prototipo evolutivo: Que no es más que realizar evoluciones sobre la base del mismo prototipo hasta determinar claramente los requerimientos.

- **Prototipo Bosquejado:** El analista de requerimientos simula las respuestas del sistema y realiza bosquejos de las interfaces de usuario y por otro lado el usuario, que es quien realiza las entradas ("utiliza el prototipo").
- **Prototipo (Tangible y Usable):** Los términos tangible y usable se refieren a desarrollar una aplicación (software) con la cual pueda interactuar como si fuera la aplicación final.

Cualquiera sea la herramienta de software que se elija utilizar para desarrollar el prototipo, se debe tener en cuenta los siguientes puntos:

Debe demandar poco esfuerzo para realizar los cambios.

Debe poseer amplia flexibilidad para el manejo de las interfaces de usuario.

Debe consumir poco tiempo para generar un nuevo prototipo (maqueta).

Documento Especificación de Requerimientos | Casos de uso.

El objetivo del documento ERS (Especificación de Requerimientos de Software) es especificar los requerimientos del sistema. En este documento se incluye una lista con los requerimientos del producto con las respectivas referencias a los documentos de todos los casos de uso que satisfacen los requerimientos.

Los requerimientos se pueden expresar de diferentes formas, desde texto sin formato estricto hasta expresiones en un lenguaje formal, pasando por todas las formas intermedias. La mayoría de los requerimientos funcionales, se pueden expresar con casos de uso. (9)

## 1.12 Diseño del Sistema.

El Diseño de sistemas es el arte de definir la arquitectura de hardware y software, componentes, módulos y datos de un sistema de cómputo para satisfacer ciertos requerimientos. Es la etapa posterior al análisis de sistemas.

El diseño de sistemas tiene un rol más respetado y crucial en la industria de procesamiento de datos. Los métodos de Análisis y diseño orientado a objetos se están volviendo en los métodos más ampliamente utilizados para el diseño de sistemas.

### 1.12.1. Conceptos del Diseño.

El diseño se centra en cuatro áreas importantes de interés: datos, arquitectura, interfaces y componentes. Es la única forma de convertir exactamente los requerimientos de un cliente en un producto o sistema de software finalizado. Entre las tareas fundamentales del diseño están: producir un diseño de datos, un diseño arquitectónico, un diseño de interfaz y un diseño de componentes, para lograr un producto con la mejor calidad posible.

### 1.12.2. Principios del Diseño.

En el libro: "*Ingeniería del Software. Un enfoque práctico*" de Pressman(5), se plantean una serie de principios básicos del diseño, entre los que se encuentran:

- El diseño deberá poderse rastrear hasta el modelo de análisis.
- El diseño no deberá inventar nada que ya esté inventado.
- El diseño deberá presentar uniformidad e integración.
- El diseño deberá estructurarse para admitir cambios.
- El diseño no es escribir código y escribir código no es diseñar.
- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminado.
- El diseño deberá revisarse para minimizar los errores conceptuales (semánticos).

### 1.12.3. Calidad del Diseño.

Para obtener productos y servicios de calidad, debemos asegurar su calidad desde el momento de su diseño. Un producto o servicio de calidad es el que satisface las necesidades del cliente, por esto, para desarrollar y lanzar un producto de calidad es necesario:

- Conocer las necesidades del cliente.

- Diseñar un producto o servicio que cubra esas necesidades.
- Realizar el producto o servicio de acuerdo al diseño.
- Conseguir realizar el producto o servicio en el mínimo tiempo y al menor coste posible.

## 1.13 Patrones de casos de uso y patrones de diseño

### 1.13.1. Patrones de Casos de Uso.

La experiencia en la utilización de casos de uso en los proyectos de los sistemas más diversos ha evolucionado en un conjunto de patrones a través de los años. A continuación una breve descripción de los patrones de casos de uso más usados (10):

- **CRUD: Completo (CRUD: Complete)**

Es un patrón de estructura, sus siglas significan crear, leer, actualizar y eliminar del inglés Create-Read-Update-Delete. Este consiste en identificar un caso de uso, llamado “Administrar Información”, que modele las diferentes operaciones que pueden ser realizadas en una misma identidad. Este patrón debe ser usado cuando todos los flujos contribuyan a un mismo valor del negocio y que sean cortos y sencillos.

- **Extensión concreta (Concrete Extension or Inclusion: Extension)**

Es un patrón de estructura y consiste en dos casos de uso y una relación de extensión relacionada entre ellos. El caso de uso extendido es concreto lo cual quiere decir que puede ser instanciado por sí mismo como también extender el caso de uso base. El patrón es aplicable cuando el flujo de uno puede extender el flujo de otro caso de uso como también ser ejecutado por sí mismo.

- **Inclusión concreta (Concrete Extension or Inclusion: Inclusion)**

Es un patrón de estructura, en este hay una relación de inclusión entre un caso de uso base al caso de uso incluido. Este último puede ser instanciado por sí mismo y el caso de uso base puede ser abstracto o concreto. Este patrón es utilizado cuando el flujo de datos de un caso de uso puede ser incluido en el flujo de otro caso de uso y ejecutarse por sí solo.

- **Reglas de negocio (Business Rules: Static Definition)**

Es un patrón de estructura. Es aplicado en casos de uso que modelan servicios que son afectados por las reglas de negocios definidas en la organización. Las reglas son descritas en un documento por separado. Su uso es apropiado cuando no hay necesidad de cambios dinámicos en las reglas de negocio.

### 1.13.2. Patrones de Diseño.

Por otra parte los patrones de diseño se adentran más en lo específico, o sea estos son utilizados en componentes y en clases individuales. Un patrón de diseño es (11):

Una solución estándar para un problema común de programación, una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios, un proyecto o estructura de implementación que logra una finalidad determinada, un lenguaje de programación de alto nivel, una manera más práctica de describir ciertos aspectos de la organización de un programa, conexiones entre componentes de programas, la forma de un diagrama de objeto o de un modelo de objeto.

Las cualidades de un patrón de diseño están definidas por su encapsulamiento, abstracción, extensión y variabilidad. Cada patrón una vez aplicado genera un contexto resultante, el cual concuerda con el contexto inicial de uno o más de uno de los patrones del catálogo siendo así la generatividad y composición otra de sus cualidades. Finalmente el equilibrio permite que cada patrón deba realizar algún tipo de balance entre sus efectos y restricciones. (11)

Los patrones se clasifican según su propósito en:

- **Patrones de Creación:** Tratan la creación de instancias o sobre qué objetos un objeto delegará responsabilidades.
    - Abstract Factory: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.
    - Builder: Permite a un objeto construir un objeto complejo especificando sólo su tipo y contenido.
    - Factory Method: Define una interfaz para crear un objeto dejando a las subclases decidir el tipo específico al que pertenecen.
    - Prototype: Permite a un objeto crear objetos personalizados sin conocer su clase exacta a los detalles de cómo crearlos.
    - Singleton: Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él.

- **Patrones Estructurales:** Tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad, describiendo así la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros.
  - Adapter: Convierte la interfaz que ofrece una clase en otra esperada por los clientes.
  - Bridge: Desacopla una abstracción de su implementación y les permite variar independientemente.
  - Composite: Permite gestionar objetos complejos e individuales de forma uniforme.
  - Decorator: Extiende la funcionalidad de un objeto dinámicante de tal modo que es transparente a sus clientes.
  - Facade: Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación.
  - Flyweight: Usa la compartición para dar soporte a un gran número de objetos de grano fino de forma eficiente.
  - Proxy: Proporciona un objeto con el que se controla el acceso a otro objeto.

- **Patrones de Comportamiento:** Tratan la interacción y cooperación entre clases. Organizan, manejan y combinan comportamientos.
  - Chain of Responsability: Evita el acoplamiento entre quien envía una petición y el receptor de la misma.
  - Command: Encapsula una petición de un comando como un objeto.
  - Interpreter: Dado un lenguaje define una representación para su gramática y permite interpretar sus sentencias.
  - Iterator: Acceso secuencial a los elementos de una colección.
  - Mediator: Define una comunicación simplificada entre clases.
  - Memento: Captura y restaura un estado interno de un objeto.
  - Observer: Una forma de notificar cambios a diferentes clases dependientes.
  - State: Modifica el comportamiento de un objeto cuando su estado interno cambia.
  - Strategy: Define una familia de algoritmos, encapsula cada uno y los hace intercambiables.
  - Template Method: Define un esqueleto de algoritmo y delega partes concretas de un algoritmo a las subclases.
  - Visitor: Representa una operación que será realizada sobre los elementos de una estructura de

objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos sobre los que opera.

Los patrones en general tienen como principal objetivo trasmitir experiencia, es de suma importancia que estos estén al alcance de los desarrolladores de software.

- **Patrones GRASP**

En diseño orientado a objetos, GRASP son patrones generales de software para asignación de responsabilidades, aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. Los diferentes patrones GRASP son (12):

- **Experto en información**

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada.

- **Creador**

El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases.

- **Controlador**

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

- **Alta cohesión**

Nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.

- **Bajo acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

## 1.14 Métricas

### 1.14.1. Métricas para la Especificación de Requerimientos.

Davis y sus colegas proponen una lista de características que pueden emplearse para valorar la Especificación de requerimientos: 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. Además, los autores apuntan que las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle.

Davis sugiere que todas puedan representarse usando una o más métricas. Por ejemplo, asumimos que hay $n_r$ requerimientos en una especificación, tal como:

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

donde $n_f$ es el número de requerimientos funcionales y $n_{nf}$ es el número de requerimientos no funcionales (por ejemplo, rendimiento).

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

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

donde $n_{ui}$ es el número de requerimientos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca de 1 está el valor de Q, menor será la ambigüedad de la especificación.

### 1.14.2. Métricas para evaluar el Diseño de Software.

La medición es esencial para cualquier disciplina de ingeniería y la ingeniería de software no es una excepción. Una métrica puede ser cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software. Éstas ayudan a la evaluación de los modelos de análisis y diseño, proporcionan una indicación de la complejidad de los diseños procedimentales y del código fuente, y ayudan a realizar pruebas más efectivas. Proporcionan al diseñador una mejor visión interna y ayudan a que el diseño evolucione a un nivel superior de calidad. Se dividen en métricas de diseño arquitectónico, métricas de diseño a nivel de componente y métricas orientadas a clases.

- **Métricas de diseño arquitectónico**

Estas se concentran en las características de la estructura del programa dándole énfasis a la estructura arquitectónica y en la eficiencia de los módulos. Estas métricas son llamadas de caja negra, ya que no requieren ningún conocimiento del trabajo interno de ningún modo en particular del sistema. Card y Glass proponen tres medidas de complejidad del software:

- Complejidad estructural. S(i), de un módulo i se define de la siguiente manera:

$$S(i) = f^2_{out}(i)$$

- Complejidad de datos. D(i) proporciona una indicación de la complejidad en la interfaz interna de un módulo i y se define como:

$$D(i) = v(i) / [f_{out}(i) + 1]$$

Donde v (i) es el número de variables de entrada y salida del módulo i.

- Complejidad de sistema. C(i), se define como la suma de las complejidades estructural y de datos, y se define como:

$$C(i)=S(i)+D(i)$$

A medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta. (5)

Por otra parte Henry y Kafura proponen la métrica de complejidad de expansión-concentración, que considera las estructuras de datos que recoge (concentra) o actualizan (expansión). También es representada una fórmula para definir esta métrica.

Fenton sugiere varias métricas de morfología simples las cuales permiten comparar diferentes arquitecturas de programa mediante un conjunto de dimensiones directas. (5)

- **Métricas de diseño a nivel de componentes**

Estas se concentran en las características internas de los componentes del software, de ahí que son llamadas métricas de caja blanca, e incluyen medidas de la cohesión, acoplamiento y complejidad del módulo. Estás medidas ayudan al desarrollador de software a juzgar la calidad de un diseño a nivel de componentes. Puede aplicarse una vez que se haya desarrollado un diseño procedimental o pueden retrasarse hasta tener disponible el código fuente. (5)

- **Métricas de cohesión**

Bieman y Ott definen una colección de métricas que se definen con cinco conceptos y medidas:

- Porción de datos. Dicho simplemente, una porción de datos es una marcha atrás a través de un módulo que busca valores de datos que afectan a la localización del módulo en el que empezó la marcha atrás. Debería resaltarse que se pueden definir tanto porciones de programas (que se centran en enunciados y condiciones) como porciones de datos.
- Símbolos léxicos (tokens) de datos. Las variables definidas para un módulo pueden definirse como señales de datos para el módulo.
- Señales de unión. El conjunto de señales de datos que se encuentran en uno o más porciones de datos.
- Señales de super-unión. Las señales de datos comunes a todas las porciones de datos de un módulo.
- Cohesión. La cohesión relativa de una señal de unión es directamente proporcional al número de porciones de datos que liga. (5)

“Todas estas métricas de cohesión tienen valores que van desde 0 a 1. Tienen un valor de 0 cuando un procedimiento tiene más de una salida y no muestra ningún atributo de cohesión indicado por una métrica particular. Un procedimiento sin señales de super-unión, sin señales comunes a todas las porciones de datos, no tiene una cohesión funcional fuerte (no hay señales de datos que contribuyan a todas las salidas). Un procedimiento sin señales de unión, es decir, sin señales comunes a más de una porción de datos (en procedimientos con más de una porción de datos), no muestra una cohesión funcional débil y ninguna adhesividad (no hay señales de datos que contribuyan a más de una salida). La cohesión

funcional fuerte y la pegajosidad se obtienen cuando las métricas de Bieman y Ott toman un valor máximo de 1." (5) Para ilustrar el carácter de estas métricas, se debe la métrica para la cohesión funcional fuerte:

$$CFF(i) = SU(SA(i))/señales\ (i)$$

Donde SU (SA (i)) denota señales de super-unión (el conjunto de señales de datos que se encuentran en todas las porciones de datos de un módulo i). Como la relación de señales de super-unión con respecto al número total de señales en un módulo i aumenta hasta un valor máximo de 1, la cohesión funcional del módulo también aumenta 1.

- **Métricas de acoplamiento**

El acoplamiento de módulo proporciona una indicación de la "conectividad" de un módulo con otros módulos, datos globales y entorno exterior. Dhama ha propuesto una métrica para el acoplamiento del módulo que combina el acoplamiento de flujo de datos y de control: acoplamiento global y acoplamiento de entorno. Las medidas necesarias para calcular el acoplamiento de módulo se definen en términos de cada uno de los tres tipos de acoplamiento apuntados anteriormente. *Para el acoplamiento de flujo de datos y de control:*

*di* = número de parámetros de datos de entrada

*ci* = número de parámetros de control de entrada

*do* = número de parámetros de datos de salida

*co* = número de parámetros de control de salida

*Para el acoplamiento global*

gd = número de variables globales usadas como datos

gc = número de variables globales usadas como control

*Para el acoplamiento de entorno:*

w = número de módulos llamados (expansión)

r = número de módulos que llaman al módulo en cuestión (concentración)

Usando estas medidas, se define un indicador de acoplamiento de módulo, *mc* de la siguiente manera:

$$m_c = k/M$$

donde *k* = 1 es una constante de proporcionalidad.

$$M = d_i + a^* c_i + d_o + b^* c_o + g_d + c^* g_c + w + r$$

*donde*:

$a=b=c=2$

Usando estas medidas, se define un indicador de acoplamiento de módulo, y cuanto mayor es el valor de este indicador, menor es el acoplamiento del módulo. (5)

- **Métricas de complejidad**

Se pueden calcular una variedad de métricas del software para determinar la complejidad del flujo de control del programa. Muchas de estas se basan en una representación denominada grafo de flujo, un grafo es una representación compuesta de nodos y enlaces (también denominados filos). Cuando se dirigen los enlaces (aristas), el grafo de flujo es un grafo dirigido.

McCabe identifica un número importante de usos para las métricas de complejidad, donde pueden emplearse para predecir información sobre la fiabilidad y mantenimiento de sistemas software, también se alimentan la información durante el proyecto de software para ayudar a controlar la actividad de diseño, en las pruebas y mantenimiento, proporcionan información sobre los módulos del software para ayudar a resaltar las áreas de inestabilidad. (5) “La métrica de McCabe proporciona una medida cuantitativa para probar la dificultad y una indicación de la fiabilidad última. Estudios experimentales indican una fuerte correlación entre la métrica de McCabe y el número de errores que existen en el código fuente, así como el tiempo requerido para encontrar y corregir dichos errores. McCabe también defiende que la complejidad ciclomática puede emplearse para proporcionar una indicación cuantitativa del tamaño máximo del módulo.”

- **Métricas de diseño de interfaz**

“Sears sugiere la conveniencia de la representación como una valiosa métrica de diseño para interfaces hombre-máquina. Una IGU(Interfaz Gráfica de Usuario) típica usa entidades de representación, íconos gráficos, texto, menús, ventanas y otras para ayudar al usuario a completar tareas. Para realizar una tarea dada usando una IGU, el usuario debe moverse de una entidad de representación a otra. Las posiciones absolutas y relativas de cada entidad de representación, la frecuencia con que se utilizan y el “costo” de la transición de una entidad de representación a la siguiente contribuirán a la conveniencia de la interfaz.” (5) Para calcular la representación óptima de una IGU, la superficie de la interfaz (el área de la pantalla) se divide en una cuadrícula. Cada cuadro de la cuadrícula representa una posible posición de una entidad de la representación.

La conveniencia de la representación es empleada para la valoración de diferentes distribuciones propuestas de IGU y la sensibilidad de una representación en particular a los cambios en las descripciones de tareas (por ejemplo, cambios en la secuencia y/o frecuencia de transiciones). Es importante apuntar que el árbitro final debería ser la respuesta del usuario basada en prototipos de IGU. Nielsen Levy afirma; que puede haber una posibilidad de éxito si se prefiere la interfaz basándose exclusivamente en la opinión del usuario ya que el rendimiento medio de tareas de usuario y su satisfacción con la IGU están altamente relacionadas. (5)

### 1.15 Conclusiones.

En el capítulo se realizó un análisis de los principales aspectos y conceptos relacionados con la migración de datos, se analizaron temas y conceptos de importancia dentro de los requerimientos y diseño de software, dejando sentadas las bases teóricas del trabajo a desarrollar, también se realizó un estudio de las posibles herramientas a utilizar.

Como resultado del estudio realizado se llegó a las siguientes conclusiones:

- Es importante la realización de la herramienta para la migración de datos en el proyecto SINAPSIS, ya que esta permitiría mantener la integridad de los datos existentes.
- Se utilizó la metodología de desarrollo RUP, por ser una guía eficiente para el desarrollo de los requerimientos y diseño del módulo Migración de Datos del proyecto SINAPSIS.
- Se aplicaron las etapas de Identificación, Análisis y Especificación de Requerimientos de la Ingeniería de Requerimientos, aplicando cada una de las actividades que estas proponen, partiendo de los procesos del negocio ya definidos, como entrada a la elicitación de requerimientos.
- Se aplicaron los patrones de diseño necesarios para no cometer errores tradicionales en el diseño del módulo Migración de Datos del proyecto SINAPSIS.
- Se utilizó UML como lenguaje de modelado, por permitir todo el modelado necesario para el diseño del módulo Migración de Datos del proyecto SINAPSIS.
- Se utilizó Visual Paradigm como herramienta CASE de desarrollo, pues permitirá el modelado de los artefactos del modelo de diseño del módulo Migración de Datos del proyecto SINAPSIS.

# Capítulo 2

## 2.1 Introducción

El presente capítulo muestra la solución escogida para obtener los requerimientos y el diseño del módulo de migración de datos del proyecto SINAPSIS. Se comienza por la aplicación de las técnicas de Ingenierías de Requerimientos para obtener las funcionalidades y restricciones del sistema .Se especifican los documentos necesarios para obtener el modelo de sistema. Se analizan los casos de usos, para modelar los artefactos necesarios para obtener el modelo de diseño.

## 2.2 Requerimientos de software

Teniendo como entrada los procesos de negocio del módulo Migración de datos, junto a la interacción y aplicando técnicas para la elicitación de requerimientos, se obtuvieron los requerimientos que debe cumplir el sistema, así como las restricciones necesarias.

Los requerimientos obtenidos persiguen llegar a un entendimiento entre el cliente y el equipo de desarrollo de las condiciones que debe presentar el producto desde el punto de vista funcional. Los mismos se agruparon en dos categorías, funcionales y no funcionales. Además de llevar un proceso de control de la calidad de la especificación de los mismos.

### 2.2.1 Funcionales.

Al aplicar las diferentes técnicas de identificación de requerimientos se obtuvieron los requerimientos funcionales. A continuación se mencionan los relacionados con el caso de uso Gestionar Proyecto, los restantes **(ver el documento de Especificación de Requerimientos Funcionales de la carpeta Artefactos)**.

- **Crear un proyecto nuevo.**

El sistema permitirá crear un nuevo proyecto.

- **Importar un proyecto existente.**

El sistema permitirá cargar un proyecto guardado anteriormente.

- **Guardar el proyecto actual.**

El sistema permitirá guardar el proyecto que se está realizando en ese momento.

### 2.2.2 No Funcionales.

A continuación se nombran algunas cualidades o propiedades que debe cumplir el sistema por cada una de las categorías, los restantes requerimientos No Funcionales **(ver documento de Especificación de Requerimientos No Funcionales de la carpeta Artefactos).**

- **Usabilidad:**

**RNF.1 Cumplir con las pautas de diseño de las interfaces.**

El sistema deberá tener una interfaz gráfica uniforme a través del mismo incluyendo pantallas, menús y opciones. Las pautas de diseño serán definidas por el equipo de diseño gráfico y se realizarán siguiendo los lineamientos de la arquitectura de información.

- **Fiabilidad:**

**RNF.2 Prever contingencias para eventos de caída del sistema.**

El sistema deberá prever contingencias que pueden afectar la prestación estable y permanente del servicio. La siguiente es la lista de las contingencias que se deben tener en cuenta y se pueden considerar críticas:

- Sobrecarga del sistema por volumen de usuarios.
- Caída del sistema por sobrecarga de procesos.
- Caída del sistema por sobrecarga de transacciones.
- Caída del sistema por volumen de datos excedido en la base o bodega de datos.

Estas consideraciones implicarán que la infraestructura técnica sobre la que se implantará el sistema garantice una alta disponibilidad del mismo.

- **Eficiencia:**

**RNF.3 Responder en tiempos las peticiones que se realicen en el sistema.**

El sistema debe ser capaz de dar respuestas a las peticiones con un nivel aceptable de desempeño. Teniendo en cuenta el nivel de concurrencia que pueda existir, debe ser capaz de prestar servicio sin que se deterioren los tiempos de respuestas.

- **Seguridad:**

**RNF.4 Permitir el intercambio de datos entre el cliente y el servidor por canales cifrados.**

El sistema deberá permitir la transmisión por canales cifrados cuando se trate de información confidencial, de manera que no viaje en texto plano por la red.

- **Reusabilidad:**

**RNF.5 Garantizar que los formatos de los archivos de salida del sistema sean compatibles con los programas más comunes.**

Los ficheros que genere el sistema deben utilizar formatos estándares como (rtf, pdf, xsl) de manera que sean compatibles con las siguientes herramientas:

- Microsoft Office 2003 o superior
- Acrobat Reader 6.0 o superior
- Open Office 2.3 o superior

- **Capacidad:**

**RNF.6 Considerar características técnicas mínimas para la ejecución en clientes**

Para que un cliente de la aplicación pueda ejecutar procesos, en línea, considerados en el sistema el punto de acceso deberá cumplir con los siguientes requerimientos mínimos.

- Procesador 2.0 GHz
- Memoria 512 MB.
- Disco duro 20 GB.
- Sistema Operativo Windows 98, 2000, XP o para Servidor o Linux.

### 2.3 Actores del sistema.

Al definir las fronteras del módulo Migración de datos se encontró el actor del sistema que interactúa con las funcionalidades del módulo. EL actor definido fue:


**Figura 4. Actor del sistema**

**Tabla _1** Descripción del actor del sistema

| **Actor** | **Descripción** |
|---|---|
| Responsable de la migración de datos. | El actor “Responsable de la Migración de Datos” es un usuario que se encargará de realizar la migración de los datos desde la base de datos del sistema Nueva Etapa a la base de datos del sistema SINAPSIS. También se encarga de crear, modificar y eliminar un proyecto, o unas reglas, flujos y las conexiones a los servidores de base de datos para así realizar todos los procesos necesarios para realizar la migración de los datos. |

### 2.4 Casos de uso del sistema.

Cada forma en que los actores usan el sistema constituye un caso de uso (CU). Luego de definidos los requerimientos del sistema y aplicando patrones como es el caso de CRUD, se obtuvieron las agrupaciones de funcionalidades que aportan resultados de valor para el actor del sistema. A continuación se enuncian los CU determinados para el módulo Migración de datos del sistema SINAPSIS.

**CU: Gestionar Proyecto**

| | |
|---|---|
| **Caso de Uso:** | Gestionar Proyecto. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita migrar los datos desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS. El CU termina cuando quedan importados en el sistema Sinapsis todos los datos. |
| **Referencias:** | RF_R.1, RF_R.2, RF_R.3 |

**CU: Establecer conexión de base de datos origen.**

| | |
|---|---|
| **Caso de Uso:** | Establecer conexión de base de datos origen. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita establecer la conexión con la base de datos del sistema Nueva Etapa, luego de que se haya creado o importado el proyecto y el Responsable de la migración de datos haya seleccionado la opción de siguiente. El CU termina cuando queda conectada la base de datos del sistema Nueva Etapa. |

| | |
|---|---|
| **Referencias:** | RF_4, RF_R.5, RF_R.6 |

**CU: Probar conexión**

| | |
|---|---|
| **Caso de Uso:** | Probar conexión. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se hayan establecido las conexiones a la base de datos del sistema Nueva Etapa y el sistema SINAPSIS. Y termina con la comprobación de dichas conexiones. |
| **Referencias:** | RF_7 |

**CU: Establecer conexión de base de datos destino**

| | |
|---|---|
| **Caso de Uso:** | Establecer conexión de base de datos destino. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita establecer la conexión con la base de datos del sistema SINAPSIS, luego de que se haya creado o importado el proyecto y el Responsable de la migración de datos haya seleccionado la opción de siguiente. El CU termina cuando queda conectada la base de datos del sistema SINAPSIS. |
| **Referencias:** | RF_R.7, RF_R.8, RF_R.9, RF_R.10 |

**CU: Poner alias a las tablas.**

| | |
|---|---|
| **Caso de Uso:** | Poner alias a las tablas. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita migrar los datos de las tablas desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS. El CU termina cuando las tablas del sistema Nueva Etapa quedan renombradas con nombres que faciliten el trabajo con las mismas. |
| **Referencias:** | RF_R.11 |

**CU: Obtener estructura de la base de datos origen.**

| | |
|---|---|
| **Caso de Uso:** | Obtener estructura de la base de datos origen. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita obtener la estructura de la base de datos origen para seleccionar los datos a migrar. El CU termina cuando se obtiene la estructura |

| | |
|---|---|
| | de la base de datos origen. |
| **Referencias:** | RF_R.12, RF_R.13, RF_R.14 |

**CU: Obtener estructura de la base de datos destino**

| | |
|---|---|
| **Caso de Uso:** | Obtener estructura de la base de datos destino. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita obtener la estructura de la base de datos destino para migrar los datos. El CU termina cuando se obtiene la estructura de la base de datos destino. |
| **Referencias:** | RF_R.13, RF_R.14, RF_R.15 |

**CU: Gestionar flujos de datos**

| | |
|---|---|
| **Caso de Uso:** | Gestionar flujos de datos. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando el Responsable de la Migración de Datos necesita gestionar un flujo de dato. Consiste en que el sistema permitirá crear flujos de datos entre la(s) tabla(s) origen y la(s) tabla(s) destino. También tiene la posibilidad de adicionar, modificar o eliminar los flujos de datos. El CU termina con la creación, modificación o eliminación de un flujo de datos. |
| **Referencias:** | RF_R.16 |

**CU: Gestionar reglas de migración de datos Campo-Campo**

| | |
|---|---|
| **Caso de Uso:** | Gestionar reglas de migración de datos Campo- Campo. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El caso de uso se inicia cuando el Responsable de la Migración de Datos necesita gestionar una regla de migración de datos campo-campo. Consiste en que el sistema permitirá crear reglas de migración campo-campo para definir los campos a transferir desde la(s) tabla(s) origen a la(s) tabla(s) destino. También tiene la posibilidad de adicionar, modificar o eliminar las reglas que ya están creadas. El caso de uso termina con la creación, modificación o eliminación de una regla. |
| **Referencias:** | RF_R.17 |

**CU: Gestionar reglas de migración para nuevo Campo Destino**

| | |
|---|---|
| **Caso de Uso:** | Gestionar reglas de migración para nuevo Campo Destino. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El caso de uso se inicia cuando el Responsable de la Migración de Datos necesita gestionar una regla de migración para un nuevo campo destino. Consiste en que el sistema permitirá crear reglas de migración de datos para nuevo campo destino y así definir los valores de los campos de la(s) tabla(s) destino que no tengan su equivalente en la(s) tabla(s) origen. También tiene la posibilidad de adicionar, modificar o eliminar las reglas que ya están creadas. El caso de uso termina con la creación, modificación o eliminación de una regla. |
| **Referencias:** | RF_R.18 |

**CU: Migrar Datos**

| | |
|---|---|
| **Caso de Uso:** | Migrar Datos. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se comienza a realizar la migración de los datos desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS el cual es el encargado de probar todas las conexiones con los servidores que tienen las bases de datos, la original y destino. El CU termina cuando se logra un una conexión con los servidores y la herramienta que se va a utilizar para realizar la migración de los datos. |
| **Referencias:** | RF_R.4, RF_R.7, RF_R.8 |

**CU: Generar notificación**

| | |
|---|---|
| **Caso de Uso:** | Generar notificación. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se comienza a realizar la migración de los datos desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS el cual es el encargado de generar todas las notificaciones que surjan durante el proceso de migración de datos de una base de dato a otra, durante este proceso puede surgir algún error con las tablas de las bases de datos, con los campos y las tuplas de dichas bases de dato. El CU termina cuando se |

| | |
|---|---|
| | genera todos los posibles errores que surjan durante el proceso de migración de datos. |
| **Referencias:** | RF_R.19, RF_R.20, RF_R.21, RF_R.22, RF_R.23, RF_R.24, RF_R.25, RF_R.26, RF_R.27, RF_R.28 |

**CU: Exportar notificación a formato PDF**

| | |
|---|---|
| **Caso de Uso:** | Exportar notificación a formato PDF. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se comienza a realizar la migración de los datos desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS y comienzan a surgir las diferentes notificaciones por algún error que pueda ocurrir en el proceso, este caso de uso va a establecer un lugar para guardar dichas notificaciones en un formato de PDF y se puedan revisar en cualquier momento que sea necesario. El CU termina cuando se logra guardar por completo las diferentes notificaciones surgidas en el proceso de migración de datos. |
| **Referencias:** | RF_R.29 |

### 2.5 Diagrama de casos de uso del sistema.

El diagrama de casos de usos del sistema (DCUS), representa la relación de los actores del sistema con los casos de usos. Para la elaboración del diagrama se aplicaron los patrones de casos de uso: extensión concreta. Finalmente quedo de la siguiente manera:

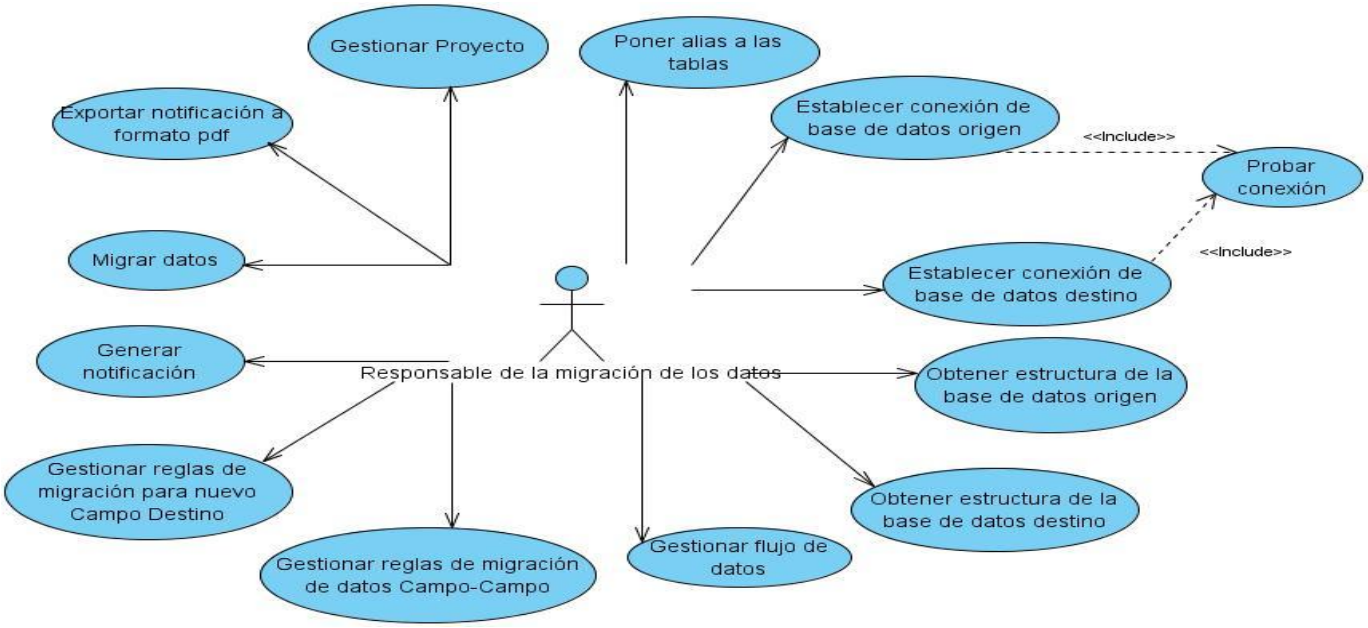



**Figura 5. Diagrama de casos de uso del sistema**

### 2.6 Descripción de casos de uso del sistema.

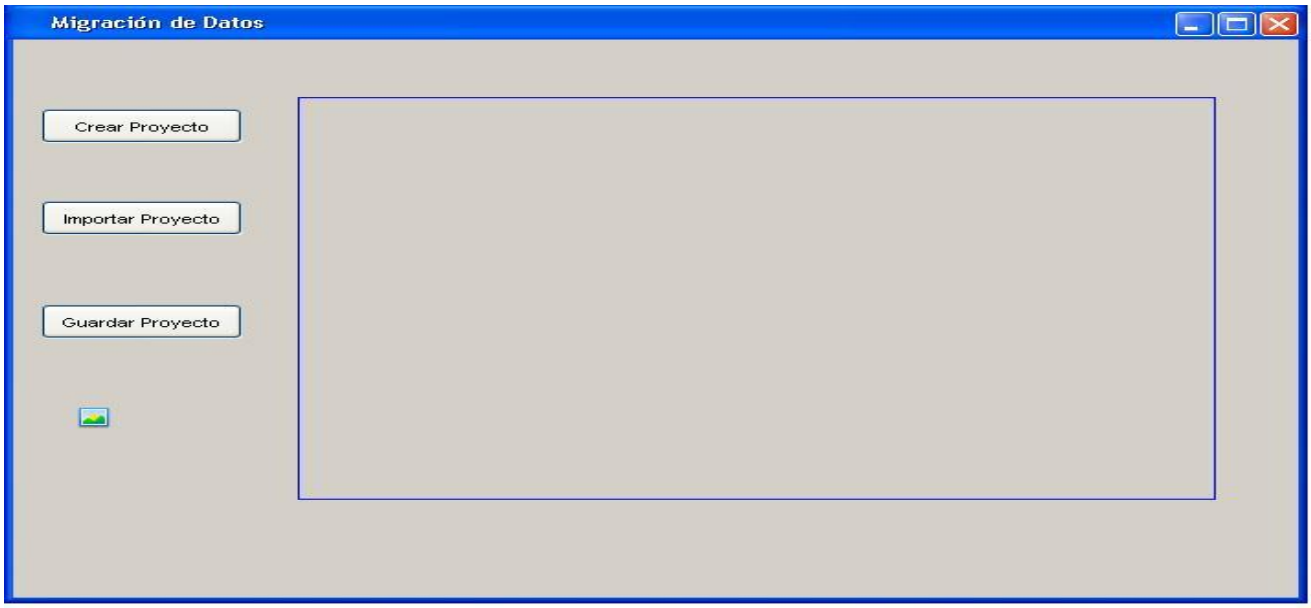
La descripción de los casos de uso constituye una guía para los desarrolladores y un documento de obligatorio cumplimiento en cuanto a desarrollo de funcionalidades en el sistema. Seguidamente se muestra la descripción de uno de los principales casos de uso del módulo Migración de datos. Para acceder a la descripción íntegra de todos los CU, ver el documento Modelo de casos de uso del sistema SINAPSIS, Módulo Migración de Datos (6).

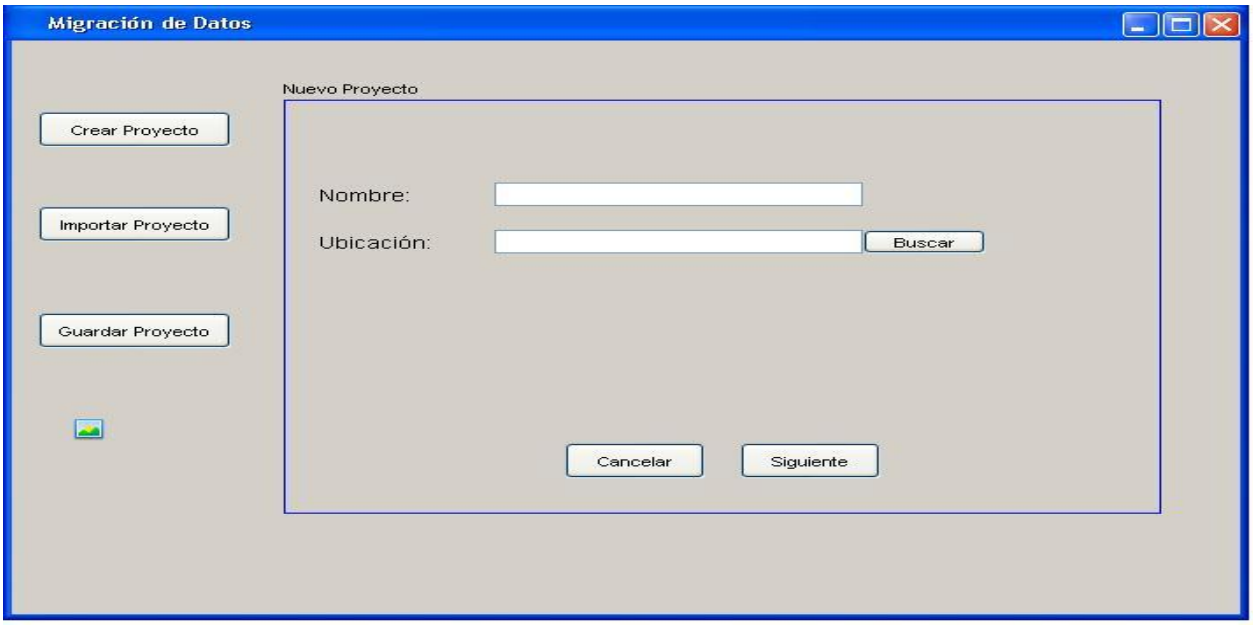
| | |
|---|---|
| **Caso de Uso:** | Gestionar Proyecto. |
| **Actores:** | Responsable de la Migración de Datos. |
| **Resumen:** | El CU se inicia cuando se necesita migrar los datos de los proyectos desde la base de datos del sistema Nueva Etapa hacia la base de datos del sistema SINAPSIS. El CU termina cuando quedan importados en el sistema SINAPSIS los datos de los proyectos. |
| **Precondiciones:** | • El sistema Nueva Etapa debe estar instalado y funcionando correctamente.<br>• El sistema SINAPSIS debe estar instalado y funcionando correctamente. |
| **Referencias** | RF_R.1, RF_R.2, RF_R.3 |
| **Prioridad** | Crítico |

| **Complejidad** | Complejo |
|---|---|
| **Nivel del caso de uso** | Usuario |

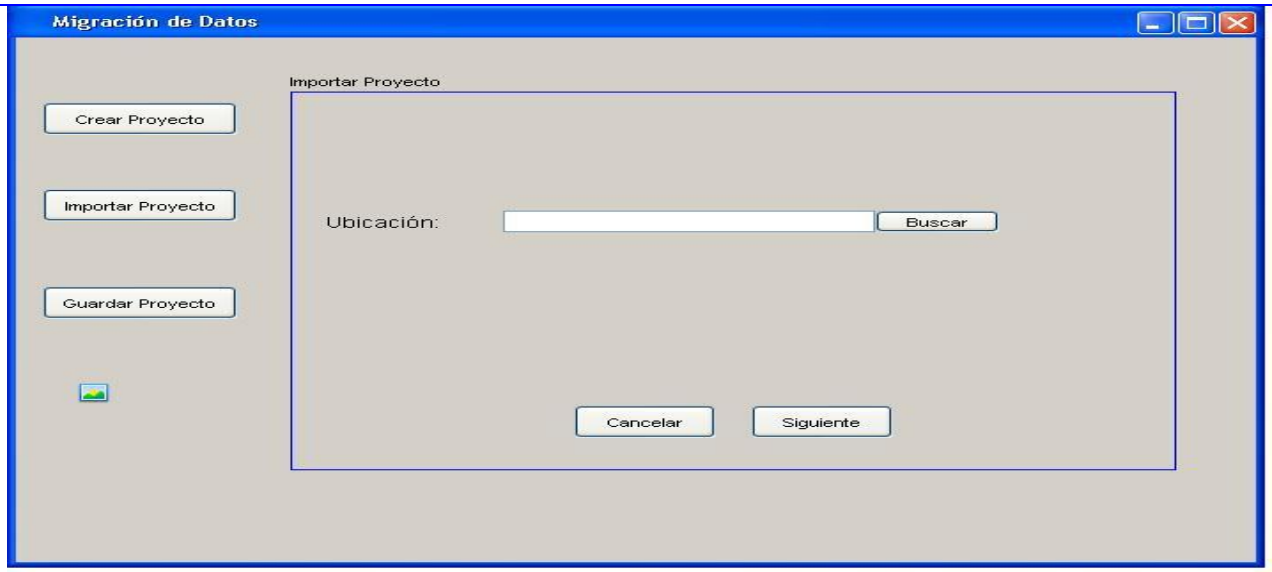
| **Flujo Normal de Eventos** | |
|---|---|
| **Acción del Actor** | **Respuesta del Sistema** |
| 1. El caso de uso inicia cuando el actor Responsable de la migración de datos selecciona la opción “Gestionar proyecto”. | 2. El sistema muestra una interfaz con las opciones de, Crear proyecto, Importar proyecto y Guardar proyecto. |
| 3. El actor Responsable de la migración de datos selecciona una de las siguientes opciones:<br>• Crear proyecto (Ver sección **“Crear proyecto”**).<br>• Importar proyecto (ver sección **“Importar proyecto”**).<br>• Guardar proyecto (ver sección **“Guardar proyecto”**). | |

***Prototipo de Interfaz***



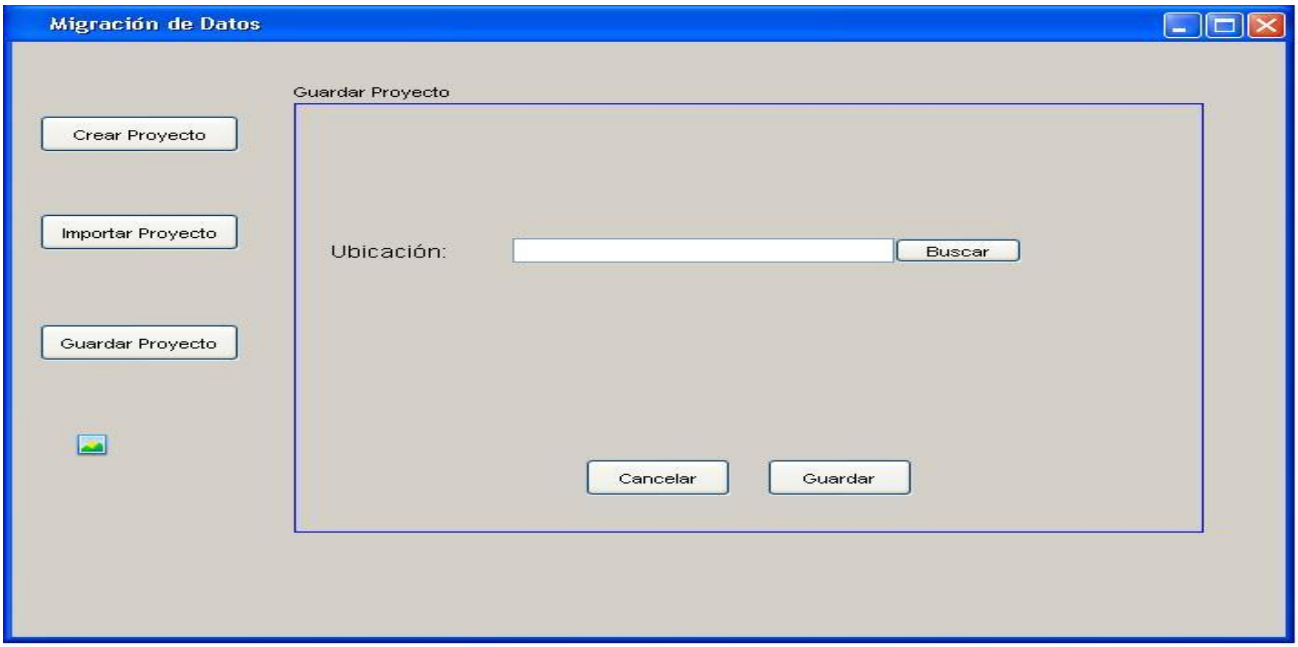
<table>
<tr><th colspan="2">Sección “Crear proyecto”</th></tr>
<tr><th>Acción del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td></td><td>1. El sistema muestra la interfaz para crear un proyecto, solicitando los siguientes datos:<br>• Nombre<br>• Ubicación</td></tr>
<tr><td>2. El actor Responsable de la migración de datos introduce los datos correspondientes y selecciona la opción “Siguiente”.</td><td>3. El sistema valida que los datos introducidos son correctos y que no hay campos obligatorios vacíos.<br>4. El sistema crea el nuevo proyecto, terminando así el caso de uso.</td></tr>
<tr><td colspan="2">Prototipo de Interfaz<br> 
</td></tr>
<tr><th colspan="2">Flujo alterno al paso 2 “Operación Siguiente”</th></tr>
<tr><th>Acción del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>2.a El actor Responsable de la migración de datos selecciona la opción “Siguiente”.</td><td>2.b El sistema pasa a la siguiente sección.</td></tr>
<tr><th colspan="2">Flujo alterno al paso 3 “Operación cancelar”</th></tr>
<tr><th>Acción del Actor</th><th>Respuesta del Sistema</th></tr>
</table>

| | |
|---|---|
| 3.a El actor Responsable de la migración de datos selecciona la opción "Cancelar". | 3.b El sistema cancela la operación y regresa al paso dos del flujo normal de eventos. |
| **Flujo alterno al paso 4 "Datos incorrectos y/o campos vacíos"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 4.a El sistema valida que los datos introducidos no son correctos y/o que hay campos obligatorios vacíos.<br>4.b El sistema muestra símbolos de error resaltando los campos obligatorios vacíos y/o donde se introdujeron datos incorrectos. Regresa al paso 1 del flujo normal de eventos de esta sección. |
| **Sección "Importar proyecto"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 1. El sistema muestra la interfaz para Importar proyecto. |
| 2. El actor Responsable de la migración de datos selecciona de donde quiere Importar el proyecto. | 3. El sistema Importa el proyecto de la dirección indicada.<br>4. El sistema actualiza el proyecto, terminando así el caso de uso. |
| ***Prototipo de Interfaz*** | |




| **Flujo alterno al paso 2 "Operación cancelar"** | |
|---|---|
| **Acción del Actor** | **Respuesta del Sistema** |
| 2.a El actor Responsable de la migración de datos selecciona la opción "Cancelar". | 2.b El sistema cancela la operación y regresa al paso dos del flujo normal de eventos. |
| **Flujo alterno al paso 3 "Operación Siguiente"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| 3.a El actor Responsable de la migración de datos selecciona la opción "Siguiente". | 3.b El sistema pasa a la siguiente sección. |
| **Sección "Guardar proyecto"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 1. El sistema muestra una interfaz para guardar el proyecto creado anteriormente o importado. |
| 2. El actor Responsable de la migración de datos selecciona la opción "Aceptar". | 3. El sistema valida que proyecto puede ser guardado (que este todo correctamente).<br>4. El sistema guarda el proyecto, terminando así el caso de uso. |

***Prototipo de Interfaz***




| **Flujo alterno al paso 2 "Operación cancelar"** | |
|---|---|
| **Acción del Actor** | **Respuesta del Sistema** |
| 2.a El actor Responsable de la migración de datos selecciona la opción "Cancelar". | 2.b El sistema cancela la operación y regresa al paso dos del flujo normal de eventos. |
| **Pos-condiciones** | ➢ El sistema queda con un nuevo proyecto creado.<br>➢ El sistema queda con un proyecto actualizado.<br>➢ El sistema queda con un proyecto guardado. |

## 2.7 Diseño

El diseño es un refinamiento del análisis que tiene en cuenta los requerimientos no funcionales, debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. En el diseño se modela el sistema y se define una arquitectura que soporte todos los requerimientos. (14)

### 2.7.1 Justificación de los Patrones de Diseño utilizados.

Entre los más evidentes en el diseño propuesto se encuentran los patrones: Experto, Creador, Bajo Acoplamiento, Alta Cohesión y Controlador.

Para contribuir a una implementación eficiente se hizo necesario el estudio de los patrones de la pandilla de los cuatro (GOF, Gang Of Four), de ellos se seleccionaron tres: Fachada y Singleton, propuestos en el

diseño inicial de la herramienta para la migración de datos en el sistema SINAPSIS y Prototype para esta iteración, los cuales se explicarán a continuación.

- **Patrón Solitario (Singleton)**

Con este patrón se garantiza una única instancia de aquellas clases que se desee tener una sola en toda la aplicación, proporcionando un punto de acceso global a dichas clases. Tiene como ventajas que reduce el espacio de nombres y es una mejora sobre las variables globales. Es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos y distintos subsistemas. Específicamente en la herramienta para la migración de datos en el sistema SINAPSI, este patrón se utiliza en las clases fachadas del sistema, garantizando una única instancia de él, esto suele hacerse cuando se aplica el Patrón Fachada, que se explicará a continuación.

- **Patrón Fachada (Facade)**

Proporciona una interfaz sencilla unificada para un conjunto de clases o subsistemas, siendo más fácil de usar. Permite reducir la complejidad y minimizar las dependencias, el acceso de los usuarios a los subsistemas es por medio de la clase fachada. Este patrón favorece a un Bajo Acoplamiento entre los usuarios y los subsistemas, respondiendo a uno de los patrones GRASP, va a permitir variar las clases internas, de manera transparente a los usuarios que las utilizan; favoreciendo de esta forma a la división en capas de la aplicación. Como se dijo anteriormente estas clases van a utilizar el patrón Solitario lo que va a permitir un acceso global a ellas.

- **Patrón Prototipo (Prototype)**

Es un patrón de creación, cuyo objetivo es especificar los tipos de objetos a crear por medio de una instancia que hace de prototipo, creando nuevos objetos copiando dicha instancia, todo se basa en clonar un prototipo dado. Cuando el responsable de la migración de datos decide realizar algunos de los procesos descritos en los CU, el sistema le muestra una interfaz de configuración, el usuario puede realizar la configuración del proceso y guardarla, puede replicar el proceso, en este caso, se crearía una copia del mismo usando el patrón Prototype y su configuración no persistiría en la Base de Datos y sólo sería visible para el Responsable de la migración de datos que lo haya replicado.

### 2.7.2 Modelo de diseño.

El Modelo de Diseño es un modelo de objetos que describe la realización de los casos de uso y al mismo tiempo constituye una abstracción del modelo de implementación y del código fuente, constituye una entrada esencial a las actividades de implementación y prueba. El presente modelo de diseño está conformado por cuatro subsistemas: Presentación, Servicios, Migración y Gestión de Proyecto.

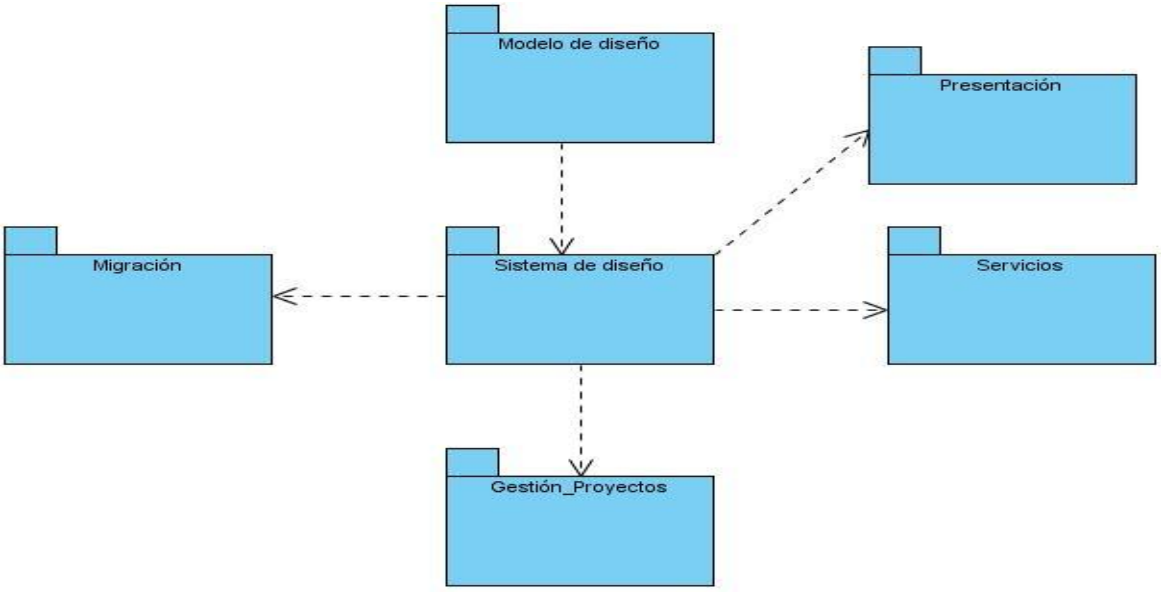



**Figura 6. Modelo de diseño.**

- **Subsistema de Presentación**

En el diseño propuesto para este subsistema solamente se encuentran aquellos componentes que permiten interactuar con el usuario, es decir formularios o ventanas para mostrar o capturar datos.

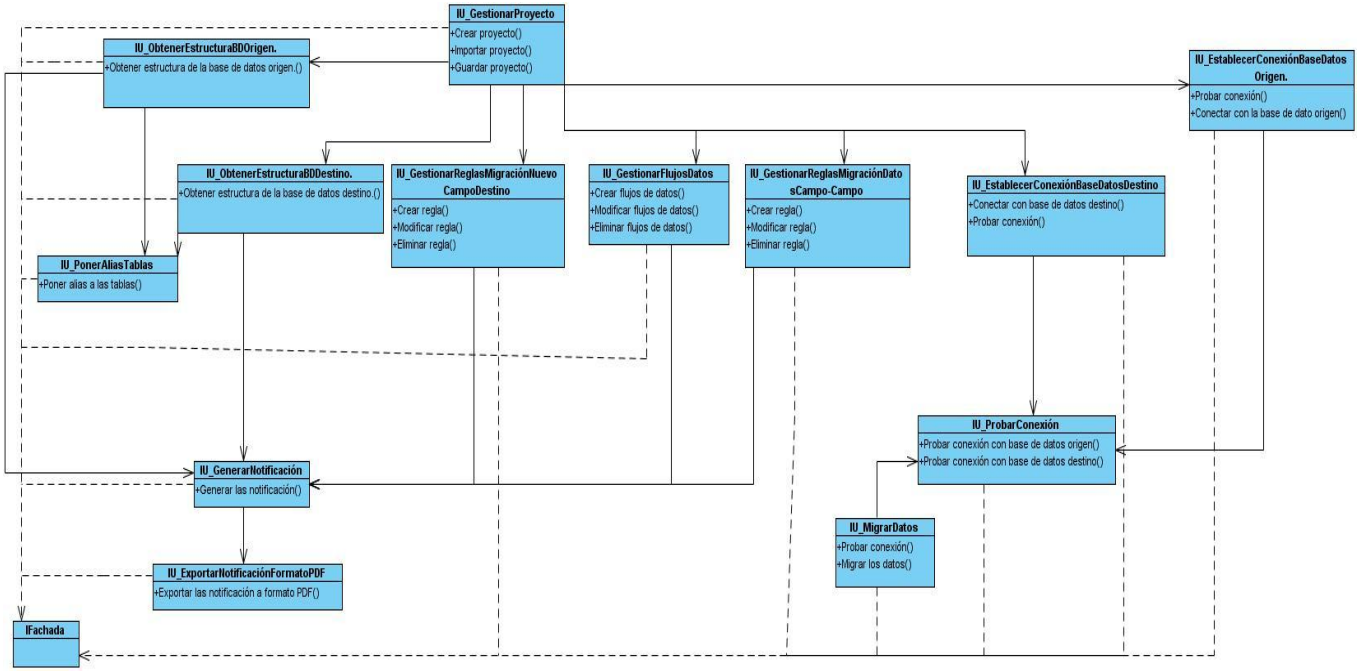



**Figura 7. Subsistema de Presentación.**

- **Subsistema de Servicios**

El diseño propuesto para este subsistema muestra aquellos componentes que son los encargados de brindar los diferentes servicios que hacen posible realizar la migración de los datos a través de las consultas y los accesos a las bases de datos.

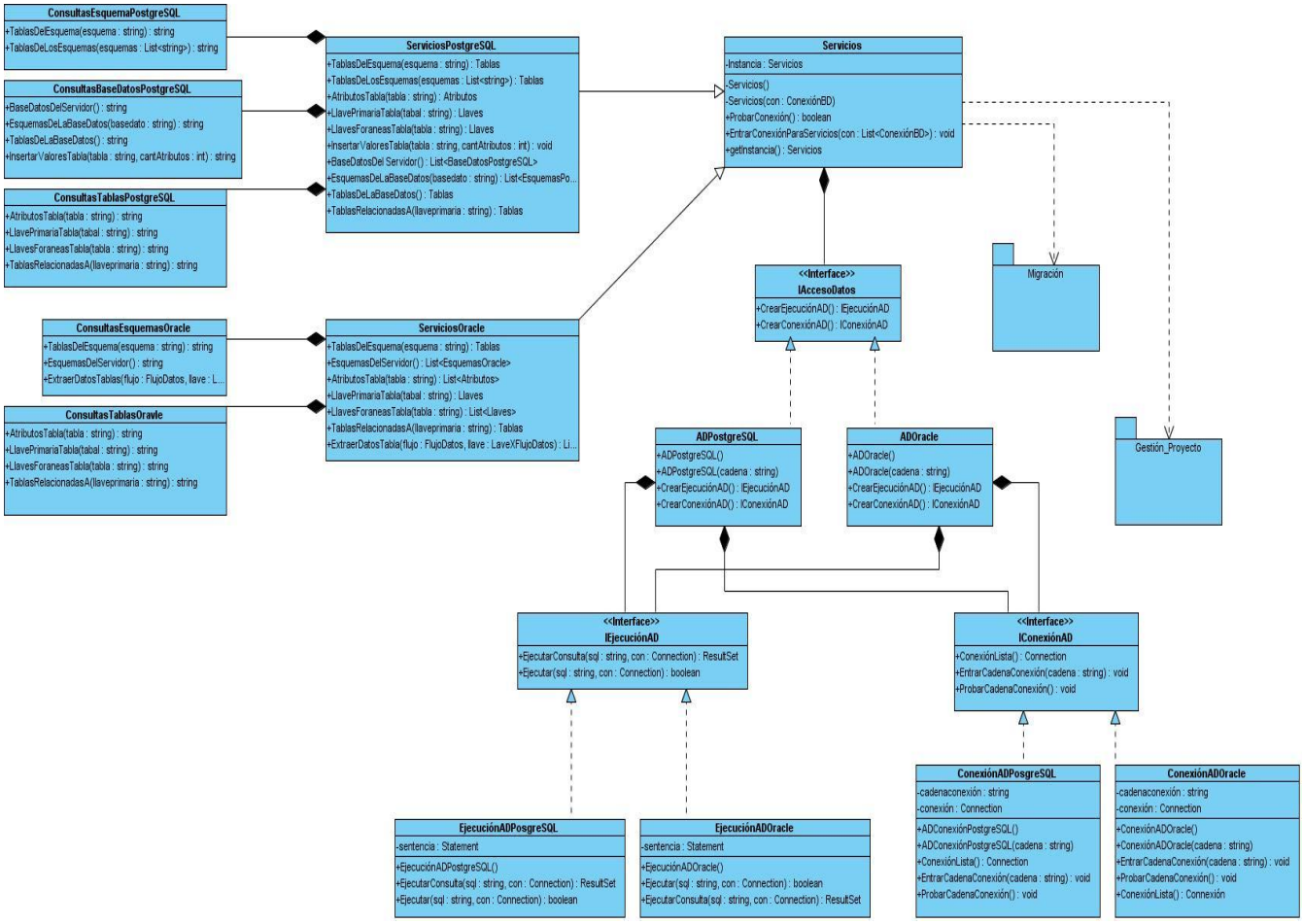



**Figura 8.Subsistema de Servicios**

- **Subsistema de Gestión de Proyectos**

En el presente diseño del subsistema se muestran todos los componentes relacionados con la gestión de los proyectos. Se identifican las relaciones establecidas entre las diferentes clases que hacen posible el funcionamiento y control de dicho subsistema.

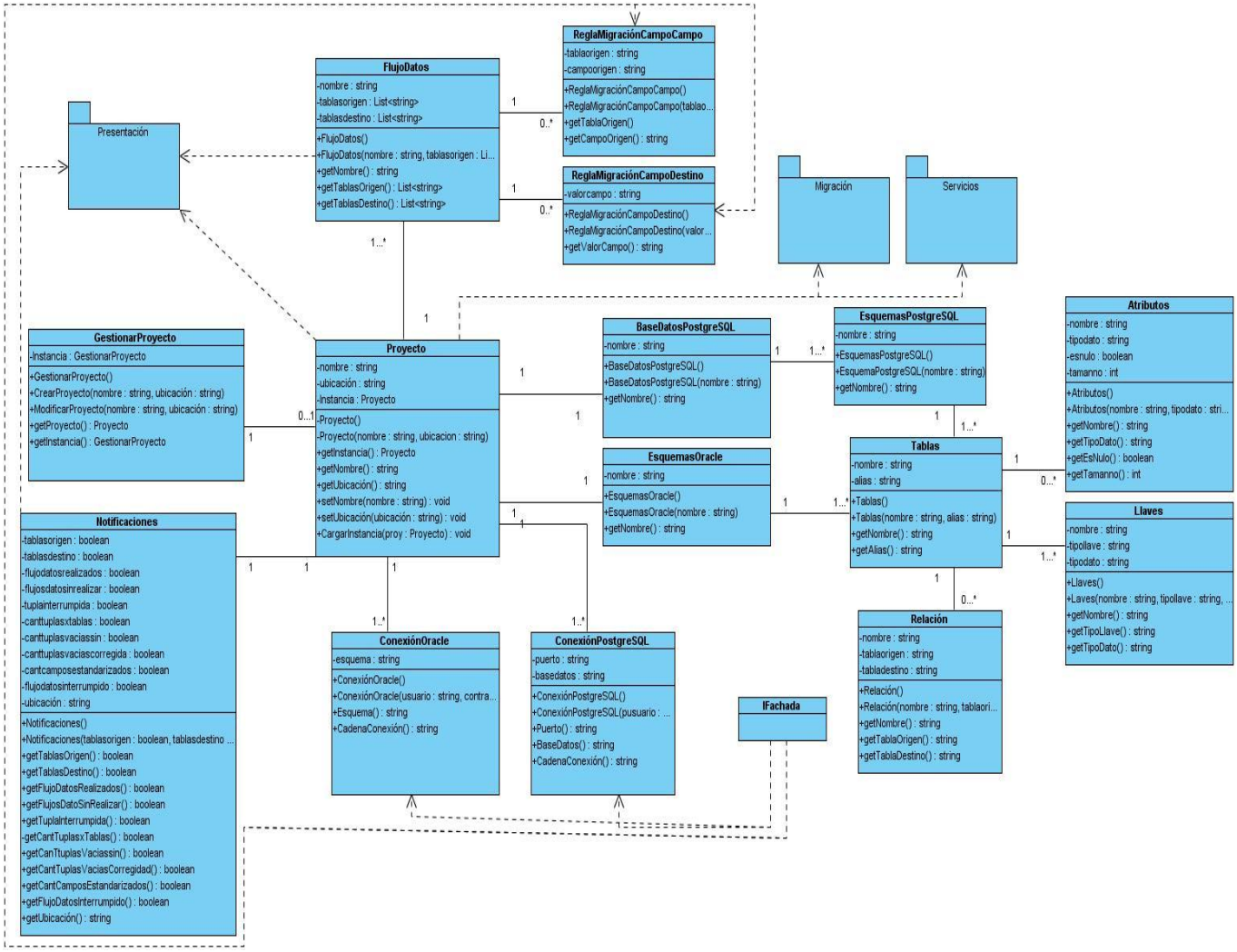



**Figura 9. Subsistema de Gestión de Proyectos**

- **Subsistema Migración**

El presente diseño muestra los componentes que hacen posible el funcionamiento del actual subsistema como acción fundamental en el desarrollo del sistema.

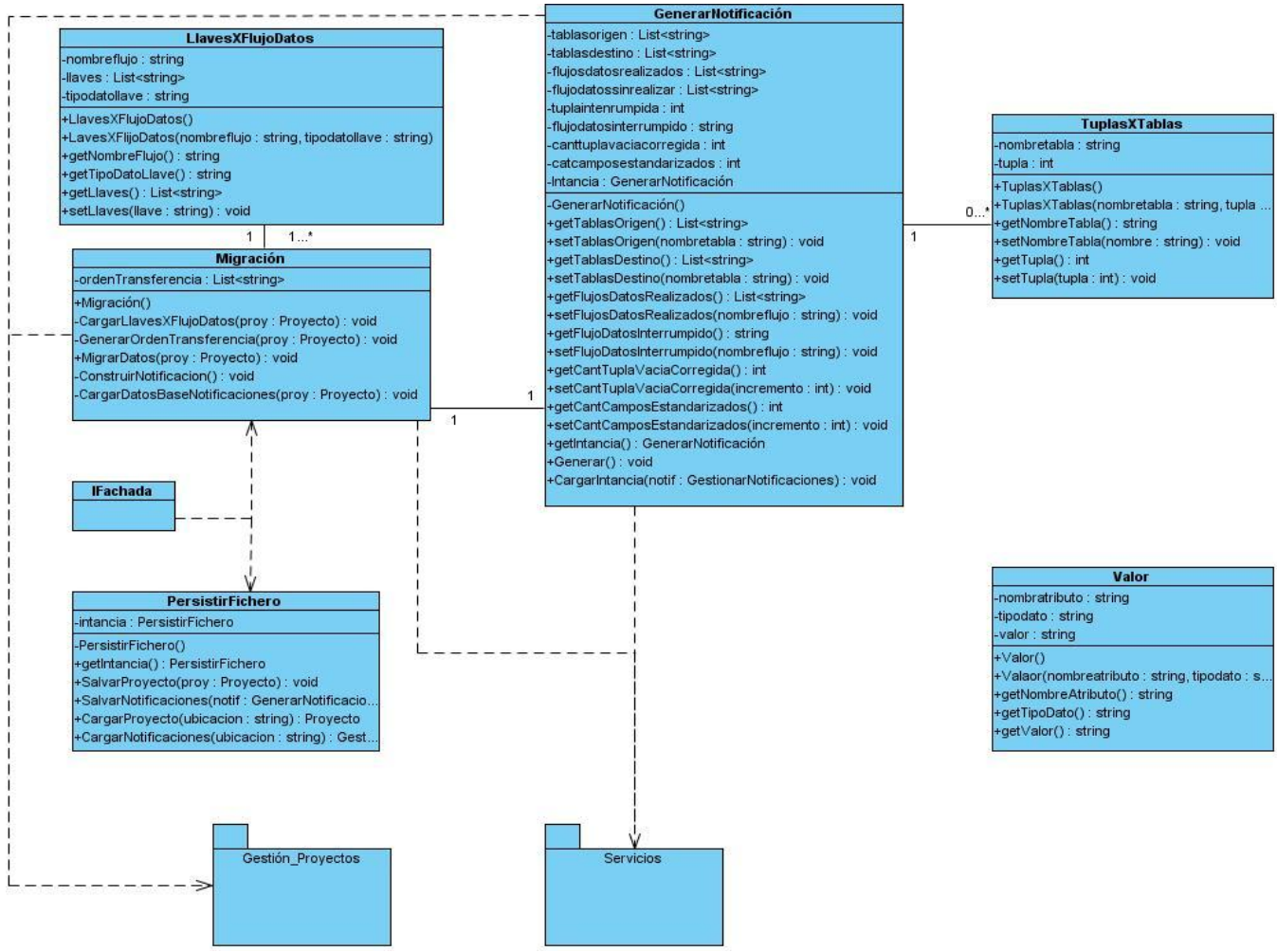



**Figura 10. Subsistema de Migración**

### 2.7.3 Diagramas de clases de diseño.

Los diagramas de clases del diseño muestran las relaciones entre clases, interfaces así como la colaboración entre ellos. Los mismos, son importantes, no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa. Para obtener el documento Modelo de Diseño íntegramente **(ver documento Modelo de Diseño de la carpeta Artefactos).**

A continuación se muestra el diagrama de clases para el CU Gestionar Proyecto:

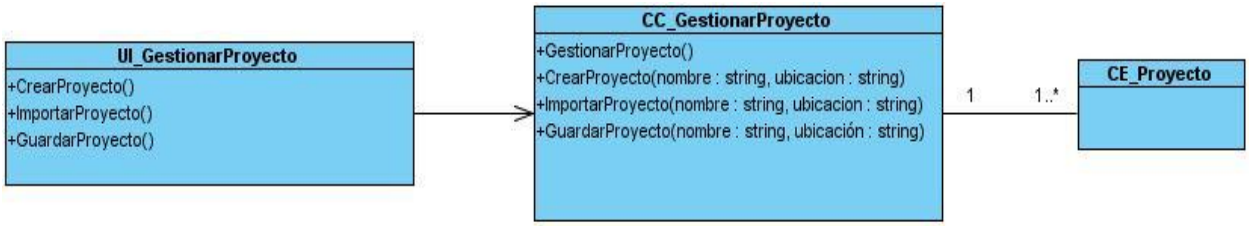



**Figura 11. Diagrama de clases del diseño: CU_Gestionar Proyecto.**

### 2.7.4 Diagramas de Secuencia.

Los diagramas de secuencia muestran una interacción ordenada según la secuencia temporal de eventos y, en particular, los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo

A continuación se muestran los diagramas de secuencia elaborados para los escenarios principales del CU Gestionar Proyecto.

Sección ““Crear proyecto”

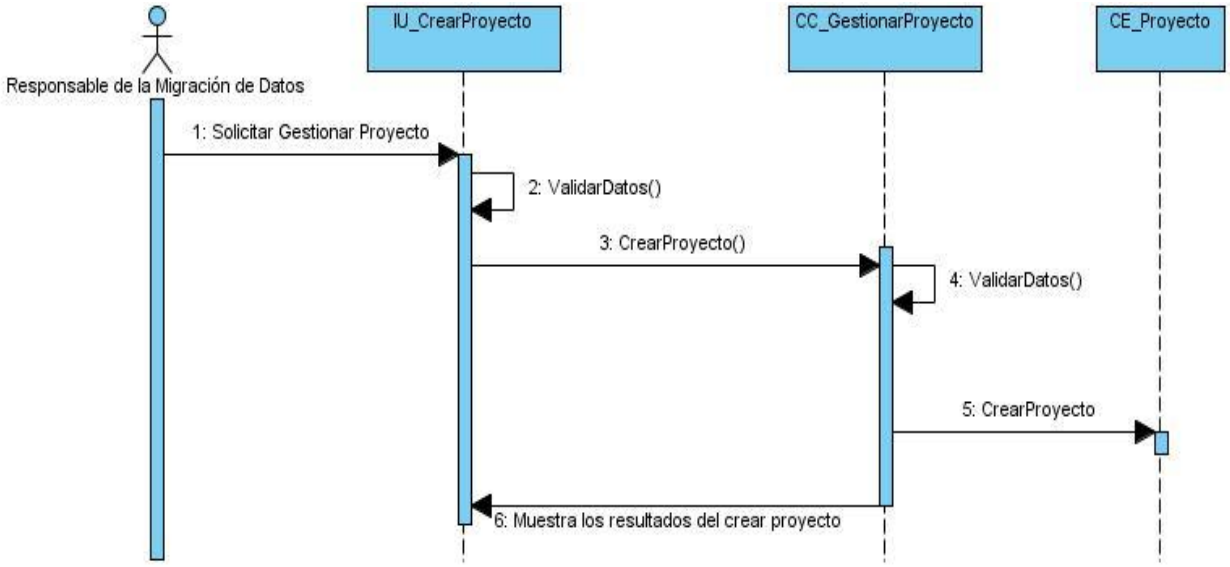



**Figura 12. Diagrama de secuencia: CU_Gestionar Proyecto, sección: “*Crear proyecto*”.**

Sección ““Importar proyecto”

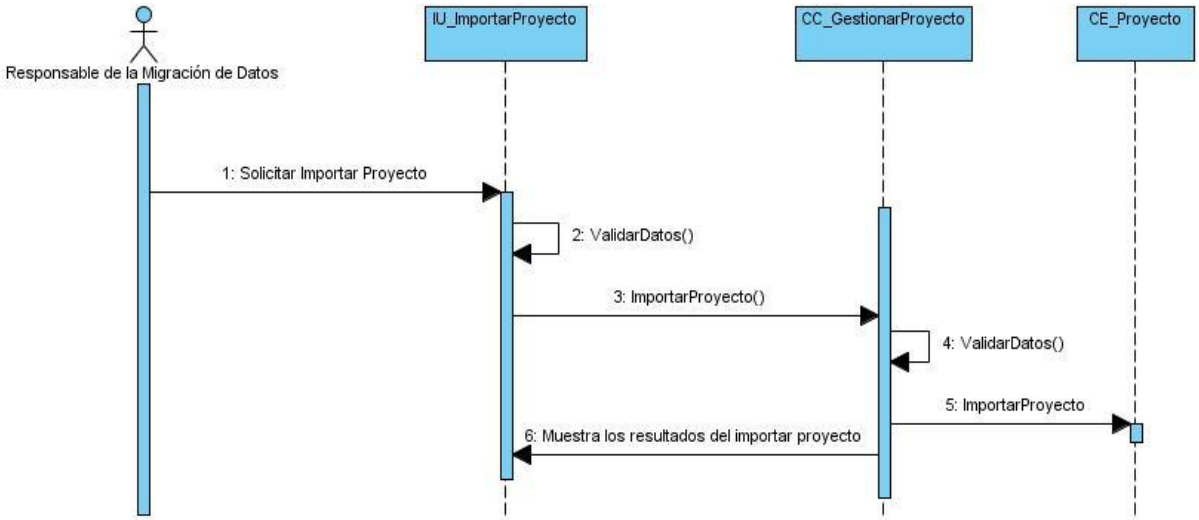



**Figura 13. Diagrama de secuencia: CU_Gestionar Proyecto, sección: “*Importar proyecto*”.**

Sección ““Guardar proyecto”

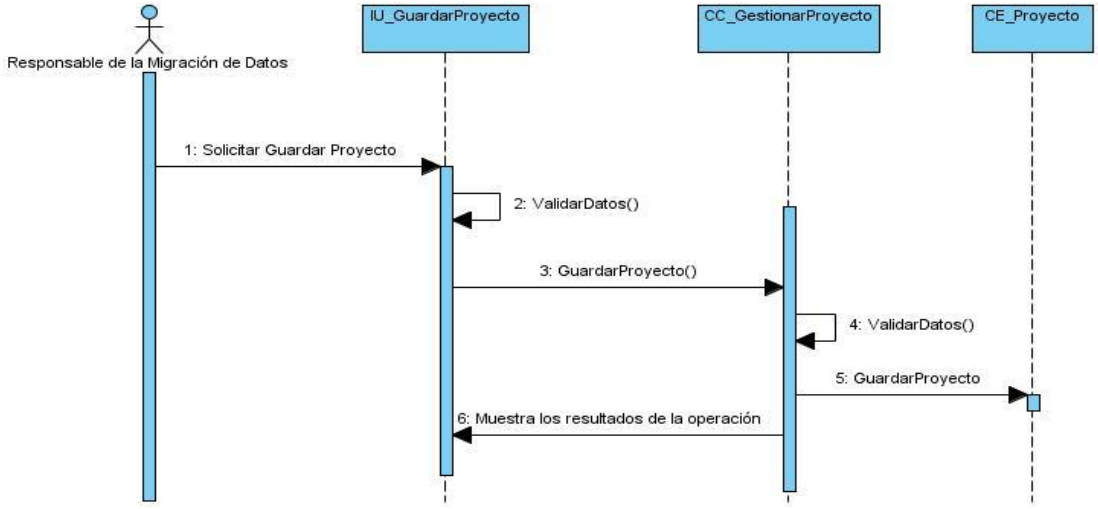



**Figura 14. Diagrama de secuencia: CU_Gestionar Proyecto, sección: *“Guardar proyecto”.***

## 2.8 Conclusiones.

Luego de tratar aspectos referentes a los requerimientos y el diseño del sistema, y obtener los principales artefactos de este flujo se puede concluir que:

- La realización de los requerimientos de forma detallada facilitó la entrada al diseño.
- Las realizaciones de los casos de uso aportaron la información necesaria para el proceso de implementación.
- El análisis de los procesos de negocio y la aplicación de técnicas para la identificación y análisis de los requerimientos, posibilitó la obtención de los resultados satisfactorios en la identificación de los requerimientos del sistema.
- La construcción del modelo de diseño permitió describir la realización física de los CU constituyendo la entrada fundamental para las actividades de implementación.

# Capítulo 3

## 3.1 Introducción

En el presente capítulo se muestran los procedimientos y métodos utilizados para medir la factibilidad de los artefactos obtenidos. Para la validación de los artefactos del modelo del sistema se muestran las listas de chequeo que se le fueron aplicados además del acta legal de liberación, mientras que para la validación de los artefactos del modelo del diseño se aplican métricas a nivel de componentes y del tamaño de clases.

## 3.2 Validación de la Especificación de Requerimientos

Para garantizar la validación de los artefactos obtenidos durante el desarrollo de la ingeniería de requerimientos fue necesario aplicar listas de chequeo y métricas a la Especificación de Requerimientos y el Modelo de Sistema, que permitieron verificar que estos artefactos cumplieran con un conjunto de características indispensables para su calidad.

### 3.2.1 Lista de Chequeo para la Especificación de Requerimientos

Las preguntas utilizadas en la lista de chequeo para la verificación de la Especificación de requerimientos (**Ver anexo 1**), se unieron en una categoría relativa a una característica de calidad. En cada una de las revisiones se fueron arreglando las no conformidades hasta que la Especificación de requerimientos satisfizo cada uno de las características de calidad contenidas en la lista.

### 3.2.2 Métricas para la Especificación de los Requerimientos

Se aplicó la métrica para la calidad de la especificación de los requerimientos de software para medir la calidad de dichos requerimientos para así saber si cumplían con la calidad requerida. Para aplicar dicha métrica, se hizo necesario realizar una lista de chequeo, aplicada en varias revisiones por parte del grupo de calidad del proyecto y de la facultad, para saber la aceptación del documento presentado con el listado de los requerimientos. A continuación se explica los pasos realizados durante este proceso:

Lo primero que se realizó fue el cálculo del número total de requerimientos.

**$R_F$**: Número de requerimientos funcionales.

**$R_{NF}$**: Número de requerimientos no funcionales.

**$R_T$**: Total de requerimientos.

$\mathbf{R_T} = \mathbf{R_F} + \mathbf{R_{NF}}$

$\mathbf{R_T} = 29 + 23$

$\mathbf{R_T} = 52$

- **Especificidad**

Para calcular la especificidad (ausencia de ambigüedad) de los requerimientos se empleó la métrica basada en la consistencia de la interpretación de los revisores para cada requerimiento. Mientras esté más cerca de 1 el valor de $\mathbf{Q_1}$ (grado de especificidad de los requerimientos), mejor será la consistencia de la interpretación de los revisores para cada requerimiento y menor será la ambigüedad de la especificación de los requerimientos.

$\mathbf{R_{ii}}$: Número de requerimientos que tuvieron interpretaciones idénticas.

$\mathbf{Q_1} = \mathbf{R_{ii}} / \mathbf{R_T}$

$\mathbf{Q_1} = 45 / 52$

$\mathbf{Q_1} = 0.86$

- **Corrección**

El cálculo del valor de corrección da una visión si los requerimientos presentan un alto nivel de corrección en la definición de los requerimientos. E igual que las otras métricas este valor debe estar lo más cerca a 1. Los pasos para calcular este valor se muestran a continuación:

$\mathbf{Q_2}$: Grado de validación de los requerimientos.

$\mathbf{R_C}$: Número de requerimientos que se han validado como correctos.

$\mathbf{R_{NV}}$: Número de requerimientos que no se han validado todavía.

$\mathbf{Q_2} = \mathbf{R_C} / (\mathbf{R_C} + \mathbf{R_{NV}})$

$\mathbf{Q_2} = 52 / (52 + 0)$

$\mathbf{Q_2} = 1$

- **Completación**

La compleción de los requerimientos funcionales se determinó de acuerdo a la compresión que tuvieron cada requerimiento por parte de las personas encargadas de realizar la revisión a dichos requerimientos. El valor óptimo de esta métrica es el más cercano a 1.

$\mathbf{R_{BC}}$: Número de requerimientos comprendidos correctamente.

$\mathbf{Q_3} = \mathbf{R_{BC}} / \mathbf{R_T}$

$\mathbf{Q_3} = 52 / 52$

$\mathbf{Q_3} = 1$

- **No redundancia**

$\mathbf{C_{RR}}$: Número de requerimientos que no se repiten.

$\mathbf{Q_5}$: Por ciento de redundancia entre 0 y 1.

$\mathbf{Q_5} = \mathbf{C_{RR}} / \mathbf{R_T}$

$\mathbf{Q_5} = 52 / 52$

$\mathbf{Q_5} = 1$

- **Consistencia interna.**

$\mathbf{C_R}$: Cantidad de requerimientos especificados.

$\mathbf{C_{RC}}$: Cantidad de requerimientos en conflicto con otros requerimientos en la especificación.

$\mathbf{Q_6}$: Por ciento de consistencia interna entre 0 y 1.

$\mathbf{Q_6} = (\mathbf{C_R} - \mathbf{C_{Rc}}) / \mathbf{C_R}$

$\mathbf{Q_6} = (52-0) / 52$

$\mathbf{Q_6} = 1$

- **Consistencia externa.**

$\mathbf{C_{RCD}}$: Cantidad de requerimientos que son consistentes con otros documentos.

**$Q_6$**: Por ciento de consistencia externa entre 0 y 1.

$$Q_6 = C_{RCD} / R_T$$

$$Q_6 = 52 / 52$$

$$Q_6 = 1$$

Para realizar un mejor análisis y entendimiento de los resultados obtenidos se realizó una gráfica que muestra los valores obtenidos:

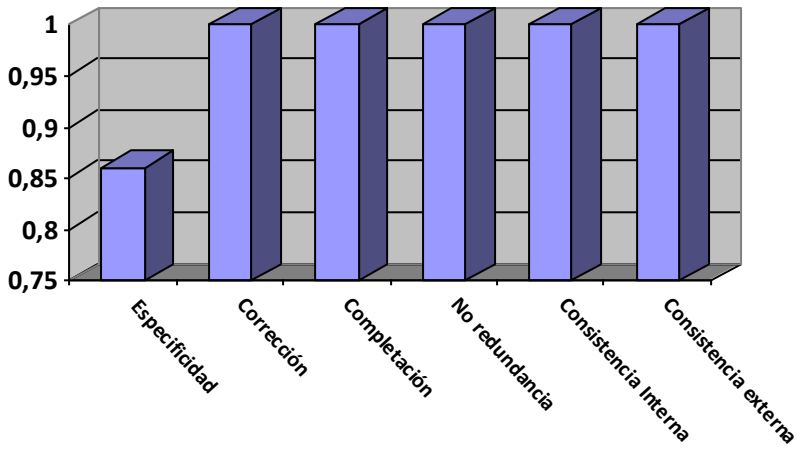



**Figura 15. Resultado de la aplicación de las métricas para la Especificación de Requerimientos.**

Después de aplicar las métricas a la Especificación de Requerimientos del módulo de Migración de Datos del proyecto Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS) se puede decir que el número de cambios realizado en los requerimientos fue bajo, el grado de aparición de ambigüedades en los requerimientos especificados es muy bajo, hay un alto nivel de corrección en la definición de los requerimientos; la compleción es alta, la redundancia de los requerimientos tiene un valor mínimo, por tanto podemos afirmar que los requerimientos cumplen con la calidad requerida.

### 3.2.3 Lista de Chequeo para el Modelo de Sistema

Para validar el Modelo de sistema se aplicó una lista de chequeo (**Ver anexo 2**), donde el objetivo fundamental es garantizar la calidad de los artefactos contenidos dentro del documento y se realizaron varias iteraciones corrigiéndose cada una de las no conformidades identificadas.

La gráfica siguiente muestra los resultados obtenidos una vez aplicada la lista de chequeo mostrando que cada artefacto cumple en un 100% los parámetros evaluados.

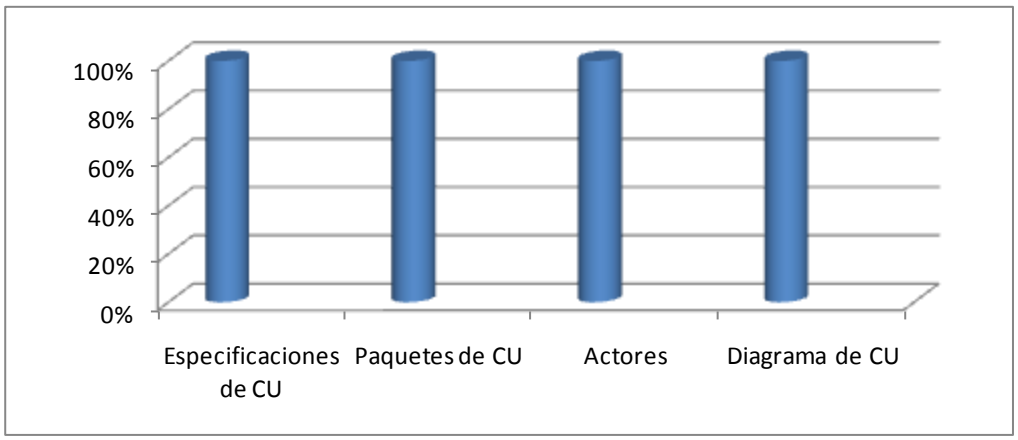



**Figura 16. Resultado de la aplicación de las listas de chequeos al Modelo de sistema.**

### 3.3 Validación del Modelo de diseño

Para obtener la validación del Modelo de diseño fue necesario aplicar un conjunto de métricas encaminadas a diferentes aspectos como son la arquitectura, los componentes y las clases garantizando la calidad del mismo.

#### 3.3.1 Métricas de Diseño Arquitectónico

- **Complejidad Estructural**

$S(i) = f_{out}^2(i)$

$S(i)$ : Complejidad estructural

$f_{out}(i)$ : Expansión del módulo Migración de Datos, que indica el número de módulos que son invocados directamente por el módulo Migración de Datos.

$S(i) = 1^2 = 1$

- **La complejidad de datos**

$$D(i) = \frac{V(i)}{f_{out}(i) + 1}$$

$D(i)$ : Complejidad en la interfaz interna del módulo Migración de Datos.

$V(i)$ : Número de variables de entrada y salida que entran y salen del módulo Migración de Datos.

$f_{out}(i)$ : Expansión del módulo Migración de Datos, que indica el número de módulos que son invocados directamente por el módulo Migración de Datos.

$$D(i) = \frac{10}{1+1} = 5$$

- **La complejidad del sistema**

$$C(i) = S(i) + D(i)$$

$C(i)$ : Complejidad del sistema.

$$C(i) = 1 + 5 = 6$$

Generalmente se proponen tres tipos de sistema según su complejidad:

- No muy complejo.
- Complejo.
- Muy complejo.

Para saber cuando un sistema está dentro de alguno de los tres grupos se proponen umbrales. Para los no muy complejos tiene que cumplirse que $D(i) <= 7$ y $S(i) <= 32$. Según esta clasificación se puede llegar a la conclusión de que el sistema no es muy complejo.

### 3.3.2 Métricas de Diseño a Nivel de Componente

- **Métricas de cohesión**

**Usabilidad de las clases.**

| Clases | Usabilidad |
|---|---|
| GestionarProyecto | 3 |
| ConexiónOracle | 1 |
| GestionarConexiónBDDestino | 1 |
| GestionarPonerAliasTablas | 2 |
| GestionarEsctucturaBDOrigen | 1 |
| GestionarEsctucturaBDDestino | 1 |
| GestionarFlujosDatos | 2 |
| GestionarReglasMigraciónDatosCampo-Campo | 2 |
| GestionarReglasMigraciónNuevoCampoDestino | 2 |
| GestionarMigrarDatos | 2 |

| | |
|---|---|
| GestionarNotificación | 3 |
| GestionarExportarNotificaicónFormatoPDF | 1 |

- **La cohesión funcional se determina con dos enfoques:**

1. Determinando la cohesión funcional fuerte (CFF) y la pegajosidad: se obtienen cuando el resultado de la métrica es de 1.

   Se define como:

   $CFF = número\ de\ súper\ adhesivos\ (i)\ /\ número\ de\ elementos\ (i)$

2. Determinando la cohesión funcional débil (CFD):

   Se define como:

   $CFD = número\ de\ adhesivos\ (i) / número\ de\ elementos\ (i)$

- **Adhesivo.** Se le llamará adhesivo a un elemento que aparece en dos o más rebanadas.
- **Súper adhesivo.** Se denomina súper adhesivo a un elemento que está en todos los elementos de un módulo.

$(i)$ Se define como la muestra.

Según los datos de las clases analizadas se tiene que:

$número\ de\ elementos = 12$

$número\ de\ súper\ adhesivos\ (i) = 0$

$número\ de\ adhesivos\ (i) = 7$

$CFD = número\ de\ adhesivos\ (i)\ /\ número\ de\ elementos\ (i)$

$CFD = 7\ /\ 12$

$CFD = 0.58$

$CFF = número\ de\ súper\ adhesivos\ (i)\ /\ número\ de\ elementos\ (i)$

$CFF = 0\ /\ 12$

$CFF = 0$

La métrica de Bieman y Ott plantea que mientras más cerca están los valores de CFF y CFD de 1 mayor será la cohesión del módulo. Después de haber realizados los cálculos sobre la cohesión funcional fuerte y la cohesión funcional débil se puede llegar a la conclusión que no existe una cohesión funcional fuerte, y la cohesión funcional débil muestra un valor de $CFD = 0.58$ , es decir no está muy cerca de 1 pero representa un 58 % de fortaleza.

### 3.3.3 Métricas Orientadas a Clases

- **Tamaño de clase (TC)**

Para medir el tamaño de clase se tienen en cuenta los siguientes aspectos:

- **Total de operaciones**, ya sean las propias o las heredadas de las clases padres e interfaces que implementen.
- **Cantidad de atributos**, tanto los de ella, como lo de los padres.
- **Promedio general** de los dos anteriores para el sistema completo.

Para evaluar esta métrica es necesario conocer los umbrales. En este caso las clases se clasifican en tres grupos según su tamaño, los que se representan en la siguiente tabla junto con los umbrales seleccionados para su clasificación.

**Tamaño de clases.**

| Clases | Nº de atributos | Nº de operaciones |
|---|---|---|
| GestionarProyecto | 2 | 4 |
| ConexiónOracle | 5 | 3 |
| GestionarConexiónBDDestino | 3 | 2 |
| GestionarPonerAliasTablas | 2 | 1 |
| GestionarEsctucturaBDOrigen | 2 | 1 |
| GestionarEsctucturaBDDestino | 1 | 1 |
| GestionarFlujosDatos | 2 | 3 |
| GestionarReglasMigraciónDatosCampo-Campo | 5 | 4 |
| GestionarReglasMigraciónNuevoCampoDestino | 4 | 4 |
| GestionarMigrarDatos | 5 | 4 |
| GestionarNotificación | 10 | 1 |
| GestionarExportarNotificaicónFormatoPDF | 0 | 1 |

Se presentó un **total de 12 clases** para un **promedio de atributos de 3.41** y **un promedio de operaciones de 2.41**.

De esta forma el umbral queda con los datos mostrados a continuación:

| Umbral | Tamaño | Cantidad de Clases |
|---|---|---|
| <=20 | Pequeño | 12 |
| >20<=30 | Medio | 0 |
| >30 | Grande | 0 |

Podemos decir que el 100% de las clases diseñadas están consideradas como pequeñas, lo cual facilitara el proceso de construcción del modulo.

### 3.4 Conclusiones.

Durante el desarrollo del presente capítulo se realizó un análisis de los resultados obtenidos en el trabajo, validándose los artefactos Modelo del Sistema y el Modelo del Diseño arribándose a las siguientes conclusiones parciales:

- Las actas de aceptación y liberación por parte del equipo de calidad de la facultad demostraron que tanto los requerimientos como la especificación de los casos de uso del sistema quedaron bien definidos, especificados y sin ambigüedad, abarcando las necesidades del cliente.
- Los valores de 1 y 5 para la complejidad estructural y de datos respectivamente indicaron que el sistema no es muy complejo.
- El diseño de las clases del módulo Migración de Datos posee una cohesión funcional con un 58 % de fortaleza, lo que demuestra que sus elementos están altamente cohesionados.
- El 100 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del módulo Migración de Datos.

Las métricas aplicadas y liberación realizada por el equipo de calidad de la facultad a los elementos del diseño del módulo Migración de Datos validan la calidad del diseño elaborado.

## Conclusiones Generales

A partir de los objetivos planteados y el trabajo realizado en esta investigación donde se obtuvo los requerimientos y el diseño de una herramienta para la migración de las bases de datos en el sistema SINAPSIS se arribó a los siguientes resultados:

- El estudio del arte de las diferentes metodologías y herramientas permitieron definir cuáles eran las adecuadas a implementar dentro del desarrollo del sistema.
- El análisis de los procesos de negocio, junto a la interacción directa con los clientes del sistema permitió que se obtuvieran resultados satisfactorios en la identificación de los requerimientos que el sistema debe cumplir.
- La elaboración del Modelo de casos de uso facilitó un mayor entendimiento y un acuerdo común entre los desarrolladores y los clientes, con respecto a las funcionalidades que debe brindar el módulo Migración de Datos.
- Se diseñó una herramienta que permite la correcta y eficiente migración de las bases de datos del sistema Nueva Etapa al sistema SINAPSIS de Oracle hacia PostgreSQL.
- El análisis de los artefactos obtenidos como resultado de la especificación y diseño del módulo Migración de Datos reveló una adecuada calidad en los mismos.
- Se generaron todos los artefactos necesarios para completar la información necesaria sobre todas las actividades realizadas y lograr una mejor comprensión para la futura implementación del sistema.

## Recomendaciones

Durante el desarrollo del presente trabajo han surgido ideas que a continuación se describirán para posteriores mejoras del mismo:

- Elaborar los artefactos restantes pertenecientes al diseño, que son necesarios para continuar con el desarrollo del módulo y que no los realiza el diseñador.
- Realizar la implementación del módulo de Migración de Datos del proyecto Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS)".
- Realizar un seguimiento de los requerimientos de software durante las posteriores fases de desarrollo aplicando la Administración de requerimientos que propone la Ingeniería de Requerimientos.

**Bibliografía**

1. **Valdés, Damián Pérez.** Maestros del web. *Maestros del web.* [En línea] [Citado el: 25 de Enero de 2010.] http://www.maestrosdelweb.com.

2. **Soto, Prof Lauro.** Tecnologico. *Tecnologico.* [En línea] [Citado el: 23 de Enero de 2010.] http://www.mitecnologico.com/Main/Tecnologico.

3. **Internet.** Lazos. *Lazos.* [En línea] [Citado el: 21 de Enero de 2010.] http://www.lazos.cl.

4. Archivos para Servidores de Bases de Datos. *Archivos para Servidores de Bases de Datos.* [En línea] [Citado el: 20 de Enero de 2010.] http://mangelp.wordpress.com/category/visual-basic-net/servidores-de-bases-de-datos/.

5. **Pressman, Roger.** *Ingeniería del Software. Un Enfoque Practico.* 2005.

6. **Corporation, Rational Sofware.** *Ayuda en línea Rational Rose.* 2003.

7. **Uruguay., Facultad de Ingeniería Universidad de la Republica |.** [En línea] [Citado el: 15 de febrero de 2010.] http://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/experiencia2002/ModsGX/modelo/roles/ana.htm..

8. **Durán Toro, Amador y Bernárdez Jiménez, Beatriz.** Metodología para la Elicitación de Requisitos de Sistemas Software. [En línea] [Citado el: 14 de febrero de 2010.] http://www.lsi.us.es/~informes/lsi-2000-10.pdf..

9. **Figueroa, Pablo.** Universidad de los Andes Departamento de Ingeniería de Sistemas y Computación. [En línea] [Citado el: 14 de febrero de 2010.] http://agamenon.uniandes.edu.co/~pfiguero/soo/Magister_Patrones/intropatrones.html..

10. **García, Joaquín.** IngenieroSoftware. [En línea] [Citado el: 11 de febrero de 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php..

11. **Larman, Craig.** *UML y Patrones Introducción al Análisis y Diseño Orientado a Objetos.* Mexico : Prentice Hall. : s.n., 1999.

12. **Letelier.** Portal de Desarrollo de Software. [En línea] [Citado el: 15 de febrero de 2010.] https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc..

13. **José Escalona, María y Koch, Nora.** Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo. [En línea] [Citado el: 13 de febrero de 2010.] http://www.lsi.us.es/docs/informes/LSI-2002-4.pdf..

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

15. **Kendall, Kenneth y Kendall, Julie.** *Análisis y Diseño de Sistemas. Tercera Edición. .* Mexico : Prentice Hall. : s.n., 1997.

16. **Quatrani, Terry.** *Visual Modeling with Rational Rose 2000 and UML. .* EEUU : Addison Wesley. : s.n., 2000.

17. **1(articulo)., Internet.** Visual Paradigm. Visual Paradigm. [En línea] [Citado el: 10 de febrero de 2010.] http://www.visual-paradigm.com/..

18. **García, Joaquín.** IngenieroSoftware. [En línea] [Citado el: 9 de febrero de 2010.] http://www.ingenierosoftware.com/analisisydiseno/uml.php..

19. **Rodríguez, Armando Labrada Leyva y Jorge Bárbaro.** *Modelo de casos de uso del sistema SINAPSIS,Módolo Migración de Datos.* UCI : s.n., 201

**Anexos**

**Anexo 1** Lista de chequeo para la Especificación de Requerimientos **(ver documento Listas de Chequeo de la carpeta Artefactos)**.

**Anexo 2** Lista de chequeo para el Modelo de sistema **(ver documento Listas de Chequeo de la carpeta Artefactos)**.

**Anexo 3** Certificado de calidad de la Especificación de Requerimientos de software, el Modelo del sistema y el Modelo del diseño.

UCI
Universidad de las Ciencias Informáticas

**Acta de Liberación de Artefactos, Grupo de Calidad Centro CEGEL de la Facultad 15 de la Universidad de las Ciencias Informáticas.**

Viernes, 14 de mayo de 2010.

Luego de haber efectuado 3 iteraciones de revisiones a los artefactos: Especificación de Requisitos, Modelo de Sistema y Modelo de diseño del módulo Migración de Datos del proyecto Sinapsis del Centro CEGEL de la Facultad 15 y haberse detectado un promedio de 8 No Conformidades, se puede afirmar que se han corregido los defectos encontrados.

Firma del Asesor y Jefe del Grupo de Calidad Centro CEGEL

Ing. Raúl Velázquez Alvarez

Decanato Facultad

**Anexo 4** Acta de aceptación de SINAPSIS.

PDVSA

Caracas, 07 de Diciembre de 2009.

Señor **Juan Carlos Montané Izaguirre**
Jefe del Proyecto **"SISTEMA NACIONAL PÚBLICO PARA EL SEGUIMIENTO DE INVERSIONES Y SECTORES".**

Estimado: **Juan Carlos Montané Izaguirre**

Después de analizado los documentos:

- Especificación de requisitos de software. Módulo de Proyectos.
- Especificación de requisitos de software. Módulo de Acciones Centralizadas.
- Especificación de requisitos de software. Módulo de Seguimiento y Control.
- Especificación de requisitos de software. Módulo de Administración de Usuario.
- Especificación de requisitos de software. Módulo de Configuración.
- Especificación de requisitos de software. Módulo de Reporte.
- Especificación de requisitos de software. Módulo de Migración de Datos.
- Modelo de Sistema. Módulo de Proyectos.
- Modelo de Sistema. Módulo de Acciones Centralizadas.
- Modelo de Sistema. Módulo de Seguimiento y Control.

PETROLEOS DE VENEZUELA
Avenida Libertador, edificio Petróleos de Venezuela, Torre Este, La Campiña. Apartado Portal 61373. Caracas 1060-A. Venezuela
Teléfono: (02) 708 1111 Fax: (02) 708 3257 www.pdvsa.com

PDVSA

- Modelo de Sistema. Módulo de Administración de Usuario.
- Modelo de Sistema. Módulo de Configuración.
- Modelo de Sistema. Módulo de Reporte.
- Modelo de Sistema. Módulo de Migración de Datos.
- Arquitectura de Software.

Visión del Proyecto concretado en el **Anexo 3 al Convenio PDVSA-ALBET**, manifestamos nuestra conformidad.

Atentamente,

Saludos,

07/12/09

**Saúl Chirinos Gutiérrez**
Gerente del Centro de Servicios Comunes-Occidente.
Jefe de Proyecto Sistema Nacional Público para el
Seguimiento de Inversiones y Sectores.

PETROLEOS DE VENEZUELA
Avenida Libertador, edificio Petróleos de Venezuela, Torre Este, La Campiña. Apartado Portal 61373. Caracas 1060-A. Venezuela
Teléfono: (02) 708 1111 Fax: (02) 708 3257 www.pdvsa.com

## Glosario de Términos

**DDL:** (Data Definition Language) es un lenguaje de definición de datos.

**DML:** (Data Manipulation Language) es un lenguaje de manipulación de datos.

**SQL:** (Structured Query Language) es un lenguaje de consulta.

**SGBD:** (Sistema Gestor de Base de Datos) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

**PostgreSQL:** PostgreSQL es un sistema gestor de base de datos.

**Oracle Instant Client:** es una herramienta clienteservidor para la gestión de bases de datos.

**Actividades:** Una acción específica está compuesta por un conjunto de actividades que la suma del cumplimiento de estas le dan cumplimiento a la de mayor nivel. Estas actividades son una unidad más atómica que permite desagregar aún más la acción permitiendo así una planificación más exacta y un control más eficiente posteriormente.

**Actores**: Roles pertenecientes a los usuarios, agrupados según sus iteraciones con las funcionalidades del sistema.

**CASE:** Acrónimo de Computer Aided Software Engineering (Ingeniería de Software Asistida por Ordenador), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

**Caso de uso (CU)**: Representación de la agrupación de funcionalidades comunes. Representan un conjunto de iteraciones entre el sistema y sus actores.

**DCUS:** Diagrama de casos de uso del sistema. Representación de la relación de los CU con los actores del sistema.

**GRASP (General Responsibility Assignment Software Patterns):** Patrones generales de software para asignación de responsabilidades.

**PDF:** Acrónimo del inglés Portable Document Format (formato de documento portátil), es un formato de almacenamiento de documentos, de tipo compuesto (imagen vectorial, mapa de bits y texto). Está especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento, determinando todos los detalles de cómo va a quedar, no requiriéndose procesos anteriores de ajuste ni de maquetación.

**Plug-in:** Un complemento (o plug-in en inglés) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

**Programación Orientada a Objetos (POO):** Es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (propiedades o datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). Expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

**Requerimientos**: Condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.

**RUP:** Son las siglas de Rational Unified Process. Se trata de una metodología para describir el proceso de desarrollo de software.

**Stakeholders:** Son los interesados, todas aquellas personas u organizaciones que afectan o son afectadas por el proyecto, ya sea de forma positiva o negativa.

**UML:** Acrónimo del inglés Unified Modeling Language (Lenguaje Unificado de Modelado). Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el **OMG** (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.