UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Requisitos y Diseño del Módulo Administración del sistema SINAPSIS

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

Autor: María del Carmen Lerma Tejada
Tutor: Ing. Daylin María Fariñas González

2009-2010

Declaración de Autoría

Declaro que soy el único autor de este trabajo y reconozco a la Facultad 15 de la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

María del Carmen Lerma Tejada

_____

Firma del Autor

Daylin María Fariñas González

_____

Firma del Tutor

Dedicatoria

*A mis **padres**,*

por todo lo que

me brindaron juntos.

*A mi **abuela**,*

por su amor incondicional.

*A mi Abuelo **Tata**,*

por cuidarme y enseñarme desde pequeña.

Agradecimientos

A mi tutora por el apoyo incondicional durante todo el tiempo de tesis. Gracias!

A mi Papá por ser tan bueno todo el tiempo y quererme tanto, siempre estaré contigo. Te quiero mucho

A mi Mamá por darme todo su amor y apoyarme en cualquier circunstancia. No existe otra mamá mejor que tú.

A mi hermano del alma, mi hermano chiquito siempre estaré cuando me necesites. Eres mi alegría por eso siempre gano yo, je!

A mi abuela por ser la persona más dedicada del mundo, gracias por todos tus consejos y el amor que me diste.

A mi tía, mi mamá chiquita, por enseñarme la tabla de multiplicación cuando no quería aprendérmela. Ah y por leerte el power y encontrar un error! ☺

A mi tío Mariano por todo su cariño brindado siempre y ser tan buen hermano.

A mi tío Jorge por ser un buen tío desde que era chiquita.

A mis primos, JJ y Syren, por ser tan alegres y tantos momentos felices compartidos.

A mis compañeros de aula durante estos años en especial Arianna, Aurelio y Mayrelis por el tiempo de estudio y momentos juntos.

A mis amigos, los amigos de tres años en el pre, amigos de Camagüey, y todos los que de una forma u otra siempre me apoyaron.

A toda mi familia y amigos, Muchas gracias!

Resumen

En el presente trabajo propone la generación de los artefactos de software necesarios para una posterior implementación del módulo Administración del Sistema Nacional Público para el Seguimiento de Inversiones y Sectores. El mismo surge de la necesidad de sustituir a Nueva Etapa, sistema utilizado en la República Bolivariana de Venezuela para la gestión del presupuesto de los proyectos de la nación, que presenta insuficiencias y no cumple las expectativas de los usuarios y clientes. Para cumplir dicho propósito se efectúa un estudio de todos los elementos necesarios para realizar una correcta identificación, análisis, especificación y validación de los requisitos que debe cumplir el sistema, además se realiza el diseño de software, identificando los subsistemas, paquetes, diagramas de clases, diagramas de secuencias y modelo de datos, referente a la parte administrativa del sistema a crear. Se les aplicó métricas a los artefactos obtenidos para verificar su calidad.

Palabras Clave: Metodología, Herramientas, Requisitos de software, Artefactos de software, Diseño de software.

Índice de Figuras

INDICE

Introducción

En la República Bolivariana de Venezuela se lleva a cabo el presupuesto por proyectos en el área del sector público por disposición de su Presidente, el Comandante Hugo Rafael Chávez Frías. El presupuesto del gobierno proporciona bienes y servicios para atender las necesidades de la comunidad; cancela servicios que permiten su funcionamiento, paga a proveedores y contratistas, resuelve problemas de pasivos laborales y contractuales con sus trabajadores.

Para llevar a cabo la gestión del presupuesto por proyectos, el Ministerio del Poder Popular para la Planificación y Desarrollo, conjuntamente con el Ministerio de Economía y Finanzas y la Vicepresidencia de la República, decidieron implantar un sistema informático que facilitara el registro, seguimiento y control de todos los proyectos que formarían parte del presupuesto anual de la nación, denominado Nueva Etapa.

A través del cual se registran todos los proyectos que son formulados y planificados por todos los entes. Se puede afirmar que el mismo no satisface todas las expectativas de los clientes y usuarios en gran medida porque no fue desarrollado utilizando una metodología de desarrollo de software, no contando con la documentación necesaria que facilite su comprensión y optimización. Además presenta deficiencias en la seguridad de la información que se maneja pues cada usuario que se registra puede tomar partido sobre cada una de las funcionalidades del sistema, sin restringirle el acceso a lo que no le está permitido. Estas inconformidades están reflejadas en un documento como insuficiencias actuales referente a la parte administrativa del sistema, que presenta Nueva Etapa.

Por la necesidad que representa para la administración pública de Venezuela un sistema confiable de este tipo, los principales organismos rectores de dicho país deciden cambiar el actual software, solicitando los servicios de la empresa ALBET.SA. En conjunto deciden implantar el Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS).

Un sistema informático que sustituirá completamente el anterior y eliminará las insuficiencias que afronta Nueva Etapa. Entre los 7 módulos en que se divide se encuentra el de Administración. Este manejará todo lo referente a los usuarios, roles y permisos que serán asignados para interactuar con el sistema. Se cuenta con un Modelo de Dominio que fue presentado como primera instancia de dicho módulo.

Surge la necesidad de diseñar una aplicación que cumpla con las expectativas de los clientes, reflejando en cada uno de los requisitos sus verdaderos intereses y proporcionando un mayor nivel de seguridad en el acceso de los datos.

Problema Científico

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

Objeto de estudio

Proceso de Desarrollo de Software.

Campo de acción

Requisitos y Diseño de software.

Objetivo General

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

Tareas Investigativas

1. Elaboración del marco teórico para la justificación del trabajo.
2. Aplicación de las etapas de la Ingeniería de Requisitos: Elicitación, Análisis, Especificación y Validación.
3. Especificación de casos de uso del sistema.
4. Diseño de artefactos del sistema.
5. Validación de especificación de casos de uso del sistema.
6. Validación del diseño.

Métodos y técnicas de investigación

Métodos Científicos

Teóricos:

- Histórico – lógico: Para el estudio del estado del arte de las metodologías y herramientas de desarrollo de software y las técnicas de diseño de software más usadas.
- Modelación: Para la creación de modelos y diagramas en la elaboración de los artefactos del módulo Administración.

Empíricos:

- Entrevista: Para interactuar con los clientes e identificar los requisitos del sistema.

Resultados Esperados:

- Especificación de Requisitos de Software.
- Modelo de casos de uso del sistema.
- Modelo de diseño.

Estructura de la tesis

Para el desarrollo del trabajo se plantea la estructuración en tres capítulos, organizados como se muestra a continuación:

Capítulo 1. Fundamentación teórica.

El capítulo 1 aborda el análisis de todos los elementos en los que se sustenta la investigación. Se hace un estudio del estado actual de las metodologías, herramientas de desarrollo de software y para el modelado de prototipos de Interfaz de Usuario no funcionales. Se tratan aspectos concernientes a la Ingeniería de Requisitos como sus etapas fundamentales y algunas técnicas recomendadas. Se presentan patrones de casos de uso y de diseño, y métricas para evaluar la calidad de la especificación de requisitos y medir el diseño.

Capítulo 2. Solución propuesta.

El capítulo 2 expone la solución que se propone para la realización del módulo Administración del sistema SINAPSIS. Se aplican parte de las técnicas de ingeniería de requisitos para obtener las funcionalidades y restricciones del sistema. Se presentan los artefactos pertinentes como resultado de la elaboración del Modelo de casos de uso del sistema (MCUS) y diseño.

Capítulo 3. Análisis de los resultados.

El capítulo 3 muestra los métodos utilizados para medir la factibilidad de los artefactos obtenidos, tanto en el MCUS como en el de diseño.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

El presente capítulo aborda los elementos que permiten realizar un correcto análisis de los requisitos y diseño del sistema, sustentados en un estudio de los mismos. Se tratan aspectos concernientes a la Ingeniería de Requisitos como sus etapas fundamentales y las técnicas más recomendadas. Se ofrece un estudio de las metodologías y herramientas de desarrollo de software más exitosas en la actualidad, además de las herramientas para el modelado de prototipos de Interfaz de Usuario no funcionales. Se presentan patrones de casos de uso y de diseño, y los lenguajes de programación encaminados a la web. Finalmente se presentan las métricas para evaluar la calidad de la especificación de requisitos y medir el diseño.

1.2 Metodologías de desarrollo de software

El uso de una metodología en el proceso de desarrollo de software es un factor clave para lograr un producto con alta calidad y que cumpla las expectativas de los clientes. Las metodologías de desarrollo de software aplican un proceso disciplinado durante todo el ciclo de vida indicando en cada etapa lo que debe hacerse. No existe una metodología universal ni se puede seleccionar una mejor que otra, es necesario identificar cual se ajusta más a las características del proyecto. En este epígrafe se estudiarán tres de las metodologías más usadas en el mundo, como son: Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF). (Corral, 2007).

Rational Unified Process (RUP)

El proceso Unificado de Desarrollo de sus siglas en ingles RUP (Rational Unified Process) es un proceso de desarrollo de software, que define “quién” está haciendo “qué”, “cuándo” y “cómo” para alcanzar un determinado objetivo. Es una metodología robusta que representa uno de los procesos más generales, pues está pensado para adaptarse a una gran variedad de sistemas de software, de diferentes tamaños de proyectos y diferentes áreas de aplicación. (Jacobson, Booch, & Rumbaugh, 2000).

Se divide en 9 flujos de trabajo y 4 fases como muestra la figura 1 (Corporation, 2003):

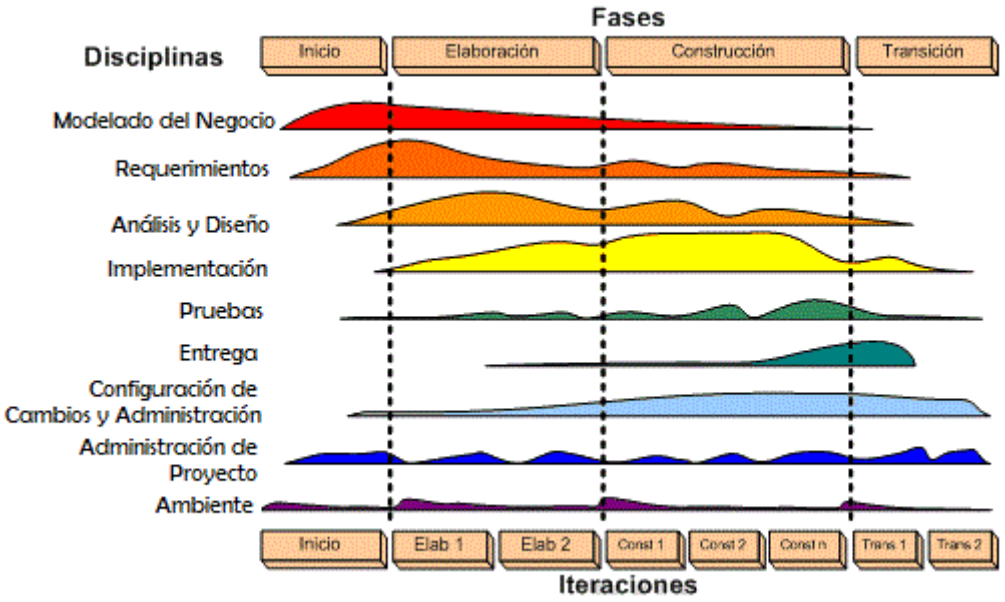


Figura 1: Fases y flujos de trabajo de RUP.

Las características que lo identifican son: dirigido por casos de uso, iterativo e incremental y centrado en la arquitectura (Jacobson, Booch, & Rumbaugh, 2000). Los casos de uso guían el proceso, que no son más que un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante, los mismos describen los requisitos de la aplicación desde el punto de vista del usuario. Se dice iterativo pues RUP propone dividir el trabajo en pequeños proyectos, cada parte más pequeña es una iteración que involucra actividades de todos los flujos de trabajo, e incremental al crecimiento del producto. Centrado en la arquitectura va de la mano con el desarrollo de los casos de uso, ambos condicionan el desarrollo del software. La arquitectura muestra la visión común del sistema completo por lo que describe los elementos del modelo que son más importantes para su construcción, es el cimiento del sistema necesario para comprenderlo, desarrollarlo y producirlo económicamente.

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 (Jacobson, Booch, & Rumbaugh, 2000).

Extreme Programming (XP)

XP es una metodología utilizada para proyectos de corto plazo y equipo. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito en el desarrollo del software. Es clasificada como ligera y se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. (Escribano, 2002) (Sanchez, 2004).

Se centra en potenciar las relaciones interpersonales promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en realimentación continua entre el cliente y el equipo de desarrollo y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes (Penadés, 2009).

Se divide en 4 fases principales (Microsoft TechNet, 2006) como muestra la siguiente figura:

Figura 2: Fases de XP.

La técnica utilizada para especificar los requisitos del software son las historias de usuario. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

El objetivo de XP es reducir el costo del cambio, usando principios, valores y prácticas básicas por medio de las cuales un proyecto debe ser más flexible al cambio (Armas, 2009). Propone empezar en pequeño y añadir nuevas características en retroalimentación continua, una de sus características es que no introduce funcionalidades antes de que sean necesarias (YAGNI: “You aren’t gonna need it” o “No lo vas a necesitar”), es decir, no hacer nada que en el futuro pueda ser útil porque probablemente no se va a necesitar y sería una pérdida de tiempo.

Microsoft Solution Framework Agile (MSF Agile)

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

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 requisitos y las estimaciones del proyecto. Además se compone de varios modelos, encargados de planificar las diferentes 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 5 fases como muestra la figura 3 (Pyme Actual, 2006), las cuales junto a los modelos mencionados anteriormente, complementan el ciclo de desarrollo de software en esta metodología.

Figura 3: Fases de MSF.

MSF se considera escalable, debido a que puede organizar equipos pequeños entre 3 ó 4 personas así como para proyectos que requieren 50 personas o más. Es flexible pues puede ser utilizada en el ambiente de desarrollo de cualquier cliente, además de ser una tecnología agnóstica la que puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

Conclusiones

Al evaluar el estudio anterior y teniendo como base las características del proyecto se adoptará RUP como metodología regente en el proceso de desarrollo de software. Además de responder satisfactoriamente ante proyectos de gran envergadura y larga duración, ha sido probada a nivel mundial y es utilizada frecuentemente en la Universidad de las Ciencias Informáticas por lo que se cuenta con el conocimiento apropiado. Permite un mantenimiento adecuado en una segunda etapa del proyecto debido a la suficiente documentación que genera, además seleccionada por contar con clientes que no tendrán una relación directa con el equipo del proyecto.

1.3 Herramientas de desarrollo de software

Las herramientas de desarrollo de software son las nombradas herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software asistida por Ordenador) que ayudan a

automatizar actividades en el proceso de desarrollo de software, así como manipular información y representar procesos de un proyecto.

Enterprise Architect, Rational Rose y Visual Paradigm son algunas de las herramientas CASE de desarrollo de software más utilizadas en el mundo de la producción de software. (Zhao, 2005)

Enterprise Architect

Enterprise Architect es una plataforma avanzada de modelado y diseño operable bajo plataforma Windows. Cubre el ciclo completo de desarrollo de software, en etapas como: captura de requisitos, análisis y diseño, pruebas y mantenimiento.

Es una herramienta gráfica multiusuario para crear sistemas robustos, que provee la especificación UML 2.1. Entre sus características se encuentra la rapidez pues carga modelos grandes en pocos segundos, proporciona una trazabilidad completa desde requisitos, análisis y diseño hasta la implementación y despliegue.

Posee generación de documentación de gran calidad y herramientas de reportes con un editor de plantilla WYSIWYG (What You See Is What You Get, lo que ves es lo que obtienes). Posibilita un modelado de procesos de negocios con extensiones personalizadas, de base de datos, así como modelar diagramas estructurales y de comportamiento.

Soporta ingeniería inversa de código de un alto rango de lenguajes de desarrollo, incluyendo Action Script, C, C++, C# y VB.NET, Java, Visual Basic 6, Python, PHP, XSD, WSDL, entre otros. Se integra a través de plug-ins con herramientas como Eclipse y Visual Estudio.Net. (Sparx Systems, 2010).

Rational Rose

Rational Rose cubre todo el ciclo de vida de un software, incluye soporte para la especificación UML 1.1. Está basada en la metodología de desarrollo RUP y utiliza un proceso de desarrollo iterativo controlado. Es un software no gratuito, solamente soportado para Windows con licencia comercial.

Dentro de las funcionalidades que soporta el Rational Rose están la 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.

Soporta una multitud IDEs y ambientes operacionales. Permite generar código a partir de un diseño en UML en distintos lenguajes de programación como: ADA, ANSI C ++, C++, CORBA, Java y Visual Basic, así como proporciona mecanismos para realizar Ingeniería Inversa. (GSInnova , 2007) (IBM).

Visual Paradigm

Visual Paradigm para UML (VP-UML) es una herramienta multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Su diseño es centrado en casos de uso, y ayuda a una rápida construcción de aplicaciones de calidad a un menor coste.

VP-UML soporta tipos los tipos de diagramas UML, así como diagramas SysML, BPMN, ERD, entre otros. Permite dibujar diagramas de casos de uso, generar código inverso y código desde diagramas en lenguajes como Java, además de documentación en formato en PDF, HTML y MS Word.

Facilita una excelente interoperabilidad mediante el intercambio de diagramas UML y modelos vía XMI. Además se puede importar diagramas creados previamente por IBM Rational Rose y ERWin, como incorporar dibujos de Visio en cualquier diagrama UML. Se integra con herramientas Java como: Eclipse, JBuilder, NetBeans/sun ONE, Weblogic Workshop, JDeveloper, IntelliJ IDE. (Visual Paradigm, 2010)**.**

Conclusiones

Se han estudiado las herramientas de desarrollo de software en cuanto a características y funcionalidades importantes para su uso como: lenguaje de modelado, plataforma y tipo de software, integración con lenguajes e IDEs, diagramas de modelado, entre otros aspectos distintivos.

A pesar de que las herramientas abordadas cuentan con características similares, se decide seleccionar Visual Paradigm. Además de ofrecer todo lo necesario para el desarrollo del sistema tiene como ventaja ser multiplataforma, asegurando su funcionamiento sobre la plataforma libre GNU/Linux, sistema operativo optado para el desarrollo del proyecto. Igualmente por su integración con el entorno de desarrollo en el que se implementará el sistema. Aunque Visual Paradigm no es una herramienta libre la Universidad cuenta con las licencias necesarias para su desarrollo.

1.4 Ingeniería de requisitos

La ingeniería de requisitos como disciplina iniciadora del proceso de desarrollo de software comprende todas las actividades relacionadas con la determinación de las necesidades o condiciones a satisfacer por un software, garantizando como principal objetivo la especificación de un sistema que cumpla con las expectativas del cliente.

Define todas las actividades involucradas en el descubrimiento, documentación y mantenimiento de los requisitos para un producto determinado (Sommerville, 2005).

Para una mejor comprensión de lo que aborda la ingeniería de requisitos es necesario primero entender el significado de requisito. Según el glosario de la IEEE se define requisito como:

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
3. Una representación documentada de una condición o capacidad como en 1. ó 2. (IEEE Standard Association, 2004).

Los requisitos del software expresan las necesidades y los apremios colocados en un producto de software que contribuye a la solución de un cierto problema del mundo real. El término ingeniería de requisitos es ampliamente utilizado para denotar la dirección sistemática de requisitos. (IEEE Swebok, 2005).

La ingeniería de requisitos facilita el mecanismo apropiado para comprender lo que quiere el cliente, analizando necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin ambigüedad, validando la especificación y gestionando los requisitos para que se transformen en un sistema operacional. (Pressman, 2005).

Etapas fundamentales

Las etapas de la Ingeniería de requisitos se encuentran claramente definidas, se estudiaran las cinco fundamentales que plantean la mayoría de los autores.

1. Elicitación. Captura o identificación de requisitos.

Este proceso tiene como principal objetivo guiar el desarrollo de software hacia el sistema correcto. Durante la identificación de los requisitos juega un papel esencial el cliente, que se convierte en un miembro más del equipo de proyecto por lo que el resultado de la captura de requisitos debe ser escrito en su lenguaje.

Esta etapa se concentra fundamentalmente en las *Técnicas de Elicitación de Requisitos* para conseguir que los stakeholders articulen sus requisitos, una actividad complicada pues los usuarios pueden tener dificultad para describir sus tareas o pueden dejar información importante sin especificar. (IEEE Swebok, 2005)

Las técnicas principales son las Entrevistas, Desarrollo conjunto de aplicaciones, Tormentas de Ideas y Casos de Uso.

Entrevistas

La entrevista es la técnica de elicitación más utilizada, es prácticamente inevitable en cualquier desarrollo por ser una de las formas de comunicación más naturales entre personas (Durán Toro,

2000). Las entrevistas le permiten al analista tener un entendimiento básico del problema y comprender los objetivos generales de la solución buscada. (Pressman, 2005)

Dentro de la realización de la entrevista se encuentran las preguntas abiertas también denominadas de libre contexto, que no responden a un 'si' o un 'no', sino que permiten una mayor comunicación y evitan la sensación de interrogatorio. Se suelen utilizar al comienzo de la entrevista pasando posteriormente a preguntas más concretas. (Durán Toro, 2000)

Desarrollo Conjunto de Aplicaciones (JAD)

La técnica Desarrollo de aplicación en conjunto, conocida por sus siglas en inglés JAD (Joint Application Development) es una alternativa a las entrevistas individuales que se desarrolla a lo largo de un conjunto de reuniones en grupo durante un periodo de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo. Se basa en cuatro principios: dinámica de grupo, el uso de ayudas visuales para mejorar la comunicación (diagramas, multimedia, herramientas CASE, etc.), mantener un proceso organizado y racional y una filosofía de documentación WYSIWYG, por lo que durante las reuniones se trabaja directamente sobre los documentos a generar. (Durán Toro, 2000).

Tormenta de Ideas (Brainstorming)

La tormenta de ideas es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Suelen realizarse por un número de cuatro a diez participantes, uno de los cuales es el jefe de la sesión, encargado de controlar la tarea. Como técnica puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de elicitación, cuando los requisitos son todavía muy difusos. (Durán Toro, 2000).

Casos de Uso

Los casos de uso son una técnica para especificar el comportamiento de un sistema. Un caso de uso es la descripción de una secuencia de interacciones entre el sistema y uno o más actores en la que se considera al sistema como una caja negra y en la que la que los actores obtienen resultados observables. Los casos de uso presentan ciertas ventajas sobre la descripción meramente textual de los requisitos funcionales y facilitan la elicitación de requisitos pues son fácilmente comprensibles por los clientes y usuarios. (Durán Toro, 2000).

2. Análisis y negociación de requisitos.

Ya una vez obtenidos los requisitos comienza entonces el proceso de análisis donde: "Los requisitos se agrupan por categorías y se organizan en subconjuntos, se estudia cada requisito en

relación con el resto, se examinan los requisitos en su consistencia, completitud y ambigüedad, y se clasifican en base a las necesidades de los clientes/usuarios". (Pressman, 2005)

Para realizar un adecuado análisis de los requisitos Pressman plantea las siguientes preguntas:

- ¿Cada requisito es consistente con objetivos generales del sistema?
- ¿Tienen todos los requisitos especificados el nivel adecuado de abstracción?
- ¿El requisito es necesario o representa una característica añadida que puede no ser esencial a la finalidad del sistema?
- ¿Cada requisito está delimitado y sin ambigüedad?
- ¿Cada requisito tiene procedencia? Es decir, existe un origen (general o específico) conocido para cada requisito.
- ¿Existen requisitos incompatibles con otros requisitos?
- ¿Es posible lograr cada requisito en el entorno técnico donde se integrará el sistema o producto?
- ¿Se puede probar el requisito una vez implementado?

Para resolver problemas de contradicción en los requisitos propuestos por los clientes se realiza un proceso de negociación con el mismo. Utilizando un procedimiento iterativo, se irán eliminando, combinando y/o modificando requisitos hasta conseguir satisfacer los objetivos propuestos.

La meta fundamental de esta etapa es solucionar cualquier contradicción o ambigüedad que existan en los requisitos expuestos por los clientes, con el objetivo de lograr requisitos concretos que cumplan con sus necesidades.

3. Especificación de requisitos.

El objetivo es esta etapa es desarrollar un documento o plantilla estándar una vez obtenido los requisitos, de manera que sean presentados de una forma más consistente y comprensible. (Pressman, 2005).

El artefacto Especificación de Requisitos de Software (ERS) es un documento que define de forma completa, precisa, y verificable los requisitos que debe cumplir el sistema tanto funcionales como no funcionales.

Una de las características fundamentales del ERS es no describir ningún detalle del diseño del software, de su verificación o de la dirección del proyecto que influyen en los requisitos, debe describir el QUÉ y no el CÓMO.

4. Validación de requisitos.

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos 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 (Pressman, 2005).

La validación representa un punto de control interno y externo; interno, porque se debe verificar internamente lo que se está haciendo, y externo, porque se debe validar con el cliente. Esta etapa realmente significa asegurarse de que el documento de especificación de requisitos represente una descripción clara del sistema, y es una verificación final de que los requisitos cubren las necesidades de los usuarios. (Garzón).

Existen varias técnicas para la validación de requisitos, como son: las revisiones técnicas formales, el prototipado y otras.

Para determinar posibles errores en los requisitos descritos Pressman plantea las siguientes preguntas:

- ¿Está el requisito claramente definido? ¿Pueden interpretarse mal?
- ¿Está identificado el origen del requisito (por ejemplo: persona, norma, documento)? ¿El planteamiento final del requisito ha sido contrastado con la fuente original?
- ¿El requisito está delimitado en términos cuantitativos?
- ¿Que otros requisitos hacen referencia al requisito estudiado? ¿Están claramente identificados por medio de una matriz de referencias cruzadas o por cualquier otro mecanismo?
- ¿El requisito incumple alguna restricción definida?
- ¿El requisito es verificable? Si es así, ¿se puede efectuar pruebas (algunas veces llamadas criterios de validación) para verificar el requisito?
- ¿Se puede seguir el requisito en el modelo del sistema que se ha desarrollado?
- ¿Se puede localizar el requisito en el conjunto de objetivos del sistema/producto?
- ¿Está el requisito asociado con rendimientos del sistema o con su comportamiento y han sido establecidas claramente sus características operacionales? ¿El requisito esta implícitamente definido?

Esta etapa es la encargada de medir la calidad de los requisitos especificados, y además de validar los errores de los mismos se comprueba que la ERS se ajuste a las necesidades del cliente.

5. Gestión de requisitos.

Los cambios en los requisitos persisten a lo largo de la vida del sistema. La gestión de requisitos incluye un conjunto de actividades que ayudan a los equipos de trabajo a identificar, controlar y seguir los requisitos y los cambios en cualquier momento (Pressman, 2005).

Para la gestión de requisitos se procede primeramente a identificar los requisitos y luego se lleva el control de estos a través de matrices de trazabilidad basadas en varios conceptos, dentro de estas se tiene:

- Matriz de seguimiento de características.
- Matriz de seguimiento de orígenes.
- Matriz de seguimiento de dependencias.
- Matriz de seguimiento de subsistemas.
- Matriz de seguimiento de interfaces.

El control de los cambios de los requisitos es importante para el buen desarrollo del sistema, por lo que es recomendable además de las actividades expuestas anteriormente para la gestión de los requisitos, tener en cuenta otras de las actividades pertenecientes a la Gestión de Configuración del Software (GCS), que son de gran utilidad para la gestión de los requisitos (Pressman, 2005).

1.5 Herramientas de modelado de Prototipos de interfaz de usuario

Existen varias herramientas de prototipos de IU no funcionales, que representan una modelo de un sistema y nos facilita verificar sus requisitos antes de comenzar la implementación del mismo. Además nos da la posibilidad de presentarlo ante clientes y el equipo de desarrollo para su aprobación. Se estudiarán algunas de las que apuntan en cuanto a sus funcionalidades para su utilización en el proyecto.

Axure

Es una herramienta de rápido prototipado y especificaciones de software, encaminada a las aplicaciones web.

Su grado de especialización es en las anotaciones, permite especificar el estado de cada elemento y el beneficio esperado, el riesgo, la estabilidad, a quién va dirigido y a quién se le asigna la tarea. Permite la realización de prototipos interactivos, es decir, se pueden simular muchas de las acciones y navegación que el usuario puede realizar en el nuevo sitio web.

La creación de documentos con especificaciones, la flexibilidad y sencillez de su uso y permitir trabajar en un mismo proyecto a varias personas a la vez son algunas de sus ventajas, mientras que el documento de especificación solamente se puede exportar en formato docx, lo que constituye su principal desventaja.

Microsoft Office Visio 2007

Microsoft Office Visio 2007 facilita la visualización, análisis y comunicación de información, sistemas y procesos complejos. Facilita y agiliza la creación de diagramas mediante plantillas, con lo que se puede mejorar la comprensión de sistemas y procesos. (Corp, Microsoft, 2006).

Dentro de la gran rama de facilidades que ofrece para distintas funciones, es una herramienta que brinda un fácil y eficiente manejo de prototipado. Los componentes que se encuentran en Windows and Dialogs y Common controls dentro de Shapes agilizan el trabajo brindando todo tipo de forma que se necesite a solo un click.

Además cuenta con distintas vistas como Drawing Explorer que en forma de árbol se accede a las carpetas para ver todos los trabajos y seleccionar directamente el deseado, Pan & Zoom permite a la vez aumentar o disminuir la imagen y saber en qué área específica del documento se está trabajando, Size & Position ofrece ver todo lo referente a tamaño y posición de un componente facilitando su alineación con otros.

La característica auto-conexión se ocupa de conectar las formas de manera automática, las distribuye de modo uniforme y las alinea con precisión. Permite además exportar a los formatos PDF y XML Paper Specification (XPS). (Microsoft Office Visio, 2009).

Visual Paradigm

La versión de Visual Paradigm 6.4 ofrece la funcionalidad para la creación de prototipos de interfaz de usuario. Contiene una variedad de componentes necesario para diseñar los prototipos. Brinda una fácil navegación por todos los diagramas, clasificados por su tipo, en la ventana Navegación de Diagramas. Además ofrece una Vista Previa al posicionar el mouse sobre algún diagrama con la posibilidad de saber con exactitud a cual nos referimos incluso antes de abrirlo. Tiene ventanas auxiliares como Descripción del diagrama que permite seleccionar con el mouse la parte del diagrama que queremos ver para comodidad del usuario.

Además de lo abordado sobre dicha herramienta en el epígrafe 1.3 sobre las herramientas de desarrollo de software y como principal razón por ser multiplataforma se seleccionó la misma para la confección de los prototipos no funcionales de la interfaz de usuario. Así se mantiene una línea de trabajo en el desarrollo del software sobre la misma herramienta.

1.6 Patrones de Casos de uso y Diseño

Un patrón "*describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que se puede utilizar un millón de veces sin tener que hacer dos veces lo mismo"* (Christopher, 1979)

Son una forma de dar solución práctica basada en la experiencia. Brindan conocimientos probados que aplicándolos ahorran tiempo y mejoran la calidad de la resolución de problemas.

Patrones de Casos de Uso

La experiencia en la utilización de casos de uso ha evolucionado en un conjunto de patrones que permiten con más precisión reflejar los requisitos reales, haciendo más fácil el trabajo con los sistemas, y mucho más simple su mantenimiento.

A continuación se describirán algunos de los principales patrones casos de uso, los de estructura como CRUD, Concordancia, Inclusión y Extensión Concreta y Múltiples actores, y Reglas del Negocio como patrón descriptivo.

Patrón CRUD (CRUD Completo)

El patrón CRUD denominado así por sus siglas Creating, Reading, Updating and Deleting es un patrón de estructura. Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual. Consta de un caso de uso, llamado ´Información CRUD´ o ´Gestionar información´, que modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico.

Existe un patrón alternativo llamado CRUD Parcial, que modela una de las vías de los casos de uso como un caso de uso separado. Es preferiblemente utilizado cuando una de las alternativas de los casos de uso es más significativa, larga o más compleja que las otras.

Patrón Concordancia

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

Dicho patrón consiste en tres casos de uso. El primero llamado “Sub-secuencia Común” que modela la secuencia de acciones que aparecen en múltiples casos de uso del modelo. Los otros dos casos de uso, que deben existir como mínimo, comparten de esta sub-secuencia común de acciones. Las variantes de este patrón son: Reuso, Adición y Especialización.

Patrón Inclusión o Extensión Concreta (Patrón de Estructura)

Este patrón contiene la relación de include o extend entre dos casos de uso, significa que se incluye explícitamente en un caso de uso base o simplemente puede o no insertarse dentro del comportamiento de dicho caso de uso.

El caso de uso base puede ser concreto o abstracto. Los casos de uso de inclusión o extensión pueden ser instanciado por sí solos.

Patrón Múltiples Actores

El patrón Múltiples Actores captura la concordancia entre actores mientras mantienen roles separados.

El caso de Rol Común plantea que cuando dos actores juegan el mismo papel hacia un caso de uso se representa otro actor, del que heredan los actores que comparten este rol. Este patrón es aplicable cuando, desde el punto de vista de un caso de uso, hay solo una entidad externa interactuando con cada instancia del caso de uso.

Mientras que Rol Diferente consiste de un caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso, o sea, interactúan de forma diferente con el caso de uso.

Patrón Reglas del Negocio

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

Este patrón es aplicado a todos los casos de uso que modelan servicios que son afectados por reglas de negocio definidas en la organización. Las reglas son descritas en un documento por separado, referenciado por la descripción del caso de uso. Es apropiado cuando no se necesita cambios dinámicos en las reglas del negocio cuando el sistema está en uso. (Övergaard & Palmkvist, 2004).

Patrones de Diseño

Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. Se adentran más en lo específico, o sea estos son utilizados en componentes y en clases individuales, su solución debe ser reutilizable para problemas de diseño similares en distintas circunstancias.

Se clasifican en dos tipos: los patrones GRASP (Patrones generales de software para asignar responsabilidades) y los GOF (Pandilla de los cuatro).

Dentro de los GRASP se encuentran los patrones Experto, Creador, Alta Cohesión, Bajo Acoplamiento, Controlador, Polimorfismo, Fabricación Pura, Indirección y No Hables con Extraños, los mismos son usados con frecuencia y representan buenas prácticas en el diseño.

Los GOF se dividen en tres conjuntos de patrones según su propósito: Patrones de Creación para la creación de instancias, Patrones Estructurales para las relaciones entre clases, combinación y formación de estructuras, y los Patrones de Comportamientos para la interacción y cooperación entre clases. Se abordarán a continuación:

Patrones de Creación:

- Abstract Factory (Fábrica abstracta): Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.
- Builder (Constructor virtual): Permite a un objeto construir un objeto complejo especificando sólo su tipo y contenido.
- Factory Method (Método de fabricación): Define una interfaz para crear un objeto dejando a las subclases decidir el tipo específico al que pertenecen.
- Prototype (Prototipo): Permite a un objeto crear objetos personalizados sin conocer su clase exacta a los detalles de cómo crearlos.
- Singleton (Instancia Única): Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él.

Patrones Estructurales:

- Adapter (Adaptador): Convierte la interfaz que ofrece una clase en otra esperada por los clientes.
- Bridge (Puente): Desacopla una abstracción de su implementación y les permite variar independientemente.
- Composite (Objeto compuesto): Permite gestionar objetos complejos e individuales de forma uniforme.
- Decorator (Envoltorio): Extiende la funcionalidad de un objeto dinámicante de tal modo que es transparente a sus clientes.
- Facade (Fachada): Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación.
- Flyweight (Peso ligero): 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:

- Chain of Responsability (Cadena de responsabilidad): Evita el acoplamiento entre quien envía una petición y el receptor de la misma.
- Command (Orden): Encapsula una petición de un comando como un objeto.
- Interpreter (Intérprete): Dado un lenguaje define una representación para su gramática y permite interpretar sus sentencias.
- Iterator (Iterador): Acceso secuencial a los elementos de una colección.
- Mediator (Mediador): Define una comunicación simplificada entre clases.
- Memento (Recuerdo): Captura y restaura un estado interno de un objeto.
- Observer (Observador): Una forma de notificar cambios a diferentes clases dependientes.

- State (Estado): Modifica el comportamiento de un objeto cuando su estado interno cambia.
- Strategy (Estrategia): Define una familia de algoritmos, encapsula cada uno y los hace intercambiables.
- Template Method (Método plantilla): define un esqueleto de algoritmo y delega partes concretas de un algoritmo a las subclases.
- Visitor (Visitante): 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. (Larman, 1999) (Gamma y varios, 1995).

1.7 Lenguajes de programación

La creación de Internet dio auge a los lenguajes de programación sobre la web de manera dinámica, para posibilitar una mayor interacción con los usuarios y dar respuestas a sus demandas. Son muchos los lenguajes que existen, tanto en lado del cliente como el servidor, se verán algunos en los que pudiera estar encaminado el desarrollo del proyecto.

C#

Es un lenguaje de programación eficaz para realizar aplicaciones empresariales, estandarizado por Microsoft y diseñado específicamente para ser utilizado en la Plataforma .NET. Toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo.

Presenta varias características muy importantes como orientado a objetos, orientado a componentes**,** ahorro de tiempo en la programación ya que posee una librería de clases muy completa y bien diseñada, y entre otras, incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, esto evita que se produzcan errores difíciles de detectar.

Las desventajas que se derivan del uso de este lenguaje de programación son que en primer lugar se tiene que conseguir una versión reciente de Visual Studio .NET, por otra parte se tiene que tener algunos requerimientos mínimos del sistema para poder trabajar adecuadamente tales como contar con Windows NT 4 o superior, tener alrededor de 4 gigas de espacio libre para su instalación, entre otros.

PHP

Hypertext Pre-processor es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. No necesita ser compilado para ejecutarse. La mayor parte de su sintaxis ha sido tomada de C,

Java y Perl con algunas características específicas. Publicado bajo la PHP License considerada como software libre, es fácil de aprender y se caracteriza por ser un lenguaje muy rápido.

Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. Soporta la mayoría de servidores Web de hoy en día y ofrece capacidad de conexión con la mayoría de los gestores de base de datos que se utilizan en la actualidad. Contiene una biblioteca nativa de funciones sumamente amplia e incluida, permite las técnicas de programación orientada a objetos, no requiere definición de tipos de variables, presenta mejoras de rendimiento y es un lenguaje multiplataforma.

Entre sus desventajas se tienen la programación orientada a objetos que aún muy deficiente para aplicaciones grandes, además de realizar todo el trabajo el servidor y no delegar al cliente por lo que puede ser más ineficiente a medida que las solicitudes aumenten de número.

Java

Java es un lenguaje de programación orientado a objetos muy extendido y popular, tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems al inicio de la década de los 90, inspira gran parte de su sintaxis en C y C++, pero tiene un modelo de objetos mucho más simple y elimina características de bajo nivel como los punteros.

Sus principales características son:

Interpretado y compilado a la vez: Es compilado, en la medida en que su código fuente se transforma en una especie de código máquina llamado bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte es interpretado ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete[1] y el sistema de ejecución en tiempo real.

Robusto: Fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Minimiza las posibilidades de errores con la comprobación de tipos, la declaración explícita de métodos y manejo de memoria como la recolección de basura (garbage collector[2]).

Dinámico: Su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas.

[1] Máquina virtual de JAVA o Java Virtual Machine (JVM). El interprete es lo que hace al lenguaje portable, multiplataforma, seguro e independiente de la arquitectura. Posibilita compilar el programa en una estación UNIX y ejecutarlo en otra con Windows.

[2] Elimina la necesidad de la liberación explícita de memoria.

Java tiene pocas desventajas que dependerá del tipo de aplicación a realizarse. Una de ellas es la velocidad, los programas hechos en Java no tienden a ser muy rápidos, como son programas interpretados nunca alcanzan la velocidad de un verdadero ejecutable. La Máquina Virtual de Java consume muchos recursos, por lo que debemos tener un ordenador con muy buenas propiedades sino notaremos mucho la recarga de memoria.

Para el desarrollo del proyecto se seleccionó Java como lenguaje de programación por las características antes mencionadas, principalmente por adquirirse gratuitamente, ser multiplataforma y tener experiencia en el mismo el equipo de proyecto.

1.8 Métricas para el diseño

Métricas de diseño arquitectónico

Las métricas de diseño arquitectónico se concentran en las características de la estructura del programa con especial énfasis a la estructura arquitectónica y en la eficiencia de los módulos. Éstas 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 en el libro *Measuring Software Design Quality* proponen tres medidas de complejidad del software:

- Complejidad estructural.
- Complejidad de datos.
- Complejidad de sistema.

A medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas. (Pressman, 2005).

Métricas de diseño a nivel de componentes

Las métricas a nivel de componentes 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. 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.

Métricas de cohesión

Bieman y Ott en *Measuring Functional Cohesion* definen una colección de métricas que proporcionan una indicación de la cohesión de un módulo. Las métricas se definen con cinco conceptos y medidas: Porción de datos, Símbolos léxicos (tokens) de datos, Señales de unión, Señales de súper-unión y Pegajosidad.
Bieman y Ott desarrollaron métricas para las cohesiones funcionales fuertes, para las cohesiones funcionales débiles y pegajosidad.

Métricas de acoplamiento

El acoplamiento del módulo proporciona una indicación de la "conectividad" de un módulo con otros módulos, datos globales y entorno exterior. Dhama en *Quantitative Models of Cohesion and Coupling in Software* 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.

Se define un indicador de acoplamiento de módulo, m_c, y cuanto mayor es el valor de este indicador, menor es el acoplamiento del módulo. Queda 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 y se calcula usando las medidas de acoplamiento: las variables para el acoplamiento de flujo de datos y de control, acoplamiento global y acoplamiento de entorno.

Métricas de complejidad

Se pueden calcular una variedad de métricas del software para determinar la complejidad del flujo de control del programa. Muchas de éstas 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.
La métrica de complejidad más ampliamente usada para el software es la complejidad ciclomática, originalmente desarrollada por Thomas McCabe. "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". ¨También defiende que la complejidad ciclomática puede emplearse para proporcionar una indicación cuantitativa del tamaño máximo del módulo. (Pressman, 2005).

Métricas de diseño de interfaz

Sears en *Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout* 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, iconos gráficos, texto, menú, 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.

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. (Pressman, 2005) (UDELAB, 2009).

Métricas orientadas a clases

La clase es la unidad principal de cualquier sistema orientado a objeto. Esto dice, que tanto las medidas y métricas para una clase individual, la jerarquía de clases y las colaboraciones de las clases resultarán de gran utilidad al ingeniero que desee estimar la calidad de un diseño.

Tamaño de clase (TC)

Las métricas orientadas al tamaño para las clases OO[3] se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo.

El tamaño general de una clase se puede determinar siguiendo los planteamientos a continuación:

- El número total de operaciones (tanto operaciones heredadas como operaciones privadas de la instancia) que están encapsuladas dentro de la clase.
- El número de atributos (tanto atributos heredados como atributos privados de la instancia) que están encapsulados en la clase.

[3] Orientado a Objetos

Si existen valores grandes de TC indican que una clase puede tener demasiada responsabilidad, lo que reduciría la reutilización de la clase y complicará la implementación y la prueba. De forma contraria sucede si los valores TC son de menor valor.

Por otra parte es necesaria una evaluación concreta de las métricas mediante los umbrales. Algunos especialistas plantean la clasificación de la siguiente manera:

Tabla 1: Clasificación de clases.

Clasificación	Valores de los umbrales
Pequeño	<= 20
Medio	>20 <= 30
Grande	>30

Finalmente se calcula los promedios correspondientes a los diferentes valores para tener una estimación general del sistema. Cuanto menor sea el valor promedio para el tamaño, será más probable que las clases dentro del sistema puedan reutilizarse.

1.9 Conclusiones

Se puede concluir que:

- Aplicando un sólido proceso de Ingeniería de Requisitos se obtienen requisitos consistentes y sin ambigüedades que satisfacen las necesidades del cliente.
- Todas las metodologías tienen sus características esenciales y no se puede definir una mejor que otra, pero es imprescindible su uso en el proceso de desarrollo de software, por lo que se selecciona según su adaptación a las características del proyecto. Se adopta RUP para el desarrollo del sistema.
- Para el desarrollo de software es de gran utilidad auxiliarse de herramientas CASE como Visual Paradigm, utilizado para modelar el DCUS, prototipos de interfaz no funcionales y elementos del diseño como subsistemas y paquetes de clases.
- Los patrones de casos de uso proveen una gran ayuda a la hora de confeccionar modelos de casos de uso.
- Los patrones de diseño facilitan diseñar correctamente en menos tiempo y evitar errores que comúnmente suele suceder en la etapa de diseño.
- Se aplican métricas para la especificación de los requisitos y el diseño con el objetivo de medir y mejorar la calidad.

Capítulo 2. Análisis de la Solución Propuesta.

2.1 Introducción

En el presente capítulo se realiza la solución propuesta para el análisis y diseño del módulo Administración del sistema SINAPSIS. Se aplican la mayoría de las etapas de la Ingeniería de Requisitos para lograr una correcta captura de requisitos con la calidad requerida. Se realiza el modelado del sistema con los artefactos pertinentes, se especifican los requisitos funcionales y no funcionales, se definen los actores, casos de uso y su descripción detallada junto con el diagrama de casos de uso del sistema. Posteriormente se analizan los elementos del sistema para obtener el modelo de diseño, que consiste en colaboraciones de clases que pueden ser agregadas en paquetes y subsistemas, para lograr una entrada adecuada a las actividades de implementación.

2.2 Requisitos de Software

Los requisitos de software quedaron definidos en el epígrafe de Ingeniería de requisitos según el estándar de la IEEE, los mismos se agrupan en Funcionales y No Funcionales. Se verán las etapas de la IR que fueron aplicadas.

Se identificaron de manera satisfactoria los requisitos a cumplir por el sistema aplicando las técnicas de elicitación de requisitos. Como primera instancia se llevó a cabo una tormenta de ideas con los clientes, con una participación libre de juicio y formalidades se obtuvo una vista general de las necesidades a automatizar por el futuro sistema. Para refinar y detallar estas primeras ideas se realizaron entrevistas directamente con los clientes, y posteriormente se utilizó la técnica JAD como suplemento a esta última.

Se clasificaron los requisitos haciendo un análisis de los mismos, en el caso de los funcionales se les asignó una prioridad y a los no funcionales se les clasificó según su tipo, además de examinarlos en cuanto a su consistencia, completitud y ambigüedad. Se especificaron en una plantilla estándar definida por el Comité de Calidad de la Universidad de las Ciencias Informáticas y fueron validados según la lista de comprobación propuesta por Pressman.

Requisitos Funcionales

Como resultado de las técnicas de identificación de requisitos aplicadas se obtuvieron las condiciones o capacidades que el sistema debe cumplir. Se les asignó una prioridad, alta o baja, según las necesidades de los clientes o usuarios. Se muestran a continuación algunos requisitos funcionales que se les dará seguimiento durante el presente trabajo. Para ver todos los requisitos

funcionales del módulo Administración referirse al Documento de Especificación de Requisitos Funcionales (Proyecto Sinapsis, 2010).

Autenticar usuario

Nombre	Valor
Especificación	El sistema permitirá autenticar a un usuario a partir de los siguientes datos: • Usuario • Contraseña
Identificador	RF.1
Prioridad	Alta

Configurar el entorno de trabajo de un usuario

Nombre	Valor
Especificación	El sistema permitirá configurar el entorno de trabajo de un usuario, habilitando y dándole acceso a las funcionalidades que tiene este usuario según los roles que le fueron asignados.
Identificador	RF.4
Prioridad	Alta

Modificar contraseña

Nombre	Valor
Especificación	El sistema permitirá modificar la contraseña de un usuario autenticado a partir de los siguientes datos: • Nueva contraseña • Confirmar contraseña
Identificador	RF.5
Prioridad	Alta

Mostrar aviso informativo del cambio de contraseña

Nombre	Valor
Especificación	El sistema permitirá mostrar un aviso confirmando el cambio de contraseña.
Identificador	RF.6
Prioridad	Media

Listar usuarios

Nombre	Valor
Especificación	El sistema permitirá listar los usuarios existentes. Los campos a mostrar serán los siguientes: • Usuario • Nombre (s) • Apellidos • Correo electrónico
Identificador	RF.9
Prioridad	Alta

Crear usuario

Nombre	Valor
Especificación	El sistema permitirá crear un usuario a partir de los siguientes datos: • Nombre persona • Primer apellido • Segundo apellido • Cédula (caracteres de tipo número) • Teléfono • Usuario • Entidad (se seleccionará la entidad a la que pertenece el usuario) • Activo (Dato que define el estado del usuario en la aplicación, el estar no activo significa que el usuario no tendrá ningún tipo de privilegio sobre el sistema, aparecerá activo por defecto) • Correo electrónico • Grupo de roles (pueden ser varios) • Rol (pueden ser varios)
Identificador	RF.10
Prioridad	Alta

Modificar usuario

Nombre	Valor
Especificación	El sistema permitirá modificar un usuario existente. Los campos a modificar serán los siguientes: • Nombre persona

	• Primer apellido • Segundo apellido • Cédula (caracteres de tipo número) • Teléfono • Entidad (se seleccionará las entidades a las que pertenece el usuario) • Usuario • Activo (Dato que define el estado del usuario en la aplicación, el estar no activo significa que el usuario no tendrá ningún tipo de privilegio en aplicación) • Correo electrónico • Grupo de roles (pueden ser varios) • Rol (pueden ser varios)
Identificador	RF.11
Prioridad	Alta

Eliminar usuario

Nombre	Valor
Especificación	El sistema permitirá eliminar un usuario existente.
Identificador	RF.12
Prioridad	Alta

Ver detalles de usuario

Nombre	Valor
Especificación	El sistema permitirá mostrar los siguientes detalles del usuario: • Nombre persona • Primer apellido • Segundo apellido • Cédula • Teléfono • Usuario • Entidad • Activo • Correo electrónico • Grupo al que pertenecerá el usuario

	• Rol asignado
Identificador	RF.13
Prioridad	Alta

Buscar usuario

Nombre	Valor
Especificación	El sistema permitirá buscar un usuario a partir de los siguientes criterios de búsqueda: • Usuario • Nombre • Cédula
Identificador	RF.14
Prioridad	Alta

Requisitos No funcionales

Las propiedades o cualidades que el producto debe tener están definidas a continuación por su tipo, se mostrará un requisito no funcional por cada clasificación, las cuales son verán Usabilidad, Fiabilidad, Eficiencia, Seguridad, Portabilidad, Capacidad y Reusabilidad. Para consultarlos todos vea el Documento de Especificación de Requisitos No Funcionales (Proyecto Sinapsis, 2010).

Cumplir con las pautas de diseño de las interfaces

Nombre	Valor
Especificación	El sistema deberá tener una interfaz gráfica uniforme a través del mismo incluyendo pantallas, menú 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.
Identificador	RNF_U.1
Tipo	Usabilidad

Prever contingencias para eventos de caída del sistema

Nombre	Valor
Especificación	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.
Identificador	RNF_F.7
Tipo	Fiabilidad

Responder en tiempos aceptables las peticiones que se realicen en el sistema

Nombre	Valor
Especificación	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.
Identificador	RNF_E.8
Tipo	Eficiencia

Permitir el intercambio de datos entre el cliente y el servidor por canales cifrados

Nombre	Valor
Especificación	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.
Identificador	RNF_S.9
Tipo	Seguridad

El sistema debe ser una aplicación Web

Nombre	Valor
Especificación	El sistema se debe ejecutar sobre un entorno web de manera que se permita su acceso desde cualquier punto del país utilizando la red.
Identificador	RNF_P.18
Tipo	Portabilidad

Definir un modelo tres capas para el sistema

Nombre	Valor
Especificación	El sistema deberá considerar en su arquitectura un modelo tres capas, donde se definen tres componentes lógicos de manera independiente: capa de presentación o interfaz de usuario, capa de lógica de negocio y capa de datos. Esta arquitectura determina una separación entre la lógica del negocio y la presentación de la aplicación, lo que permite el uso de servicios desde la capa de lógica del negocio ya sea por la capa de presentación del sistema o por otros sistemas que requieran el mismo servicio.
Identificador	RNF_R.24
Tipo	Reusabilidad

Considerar características técnicas mínimas para la ejecución en clientes

Nombre	Valor
Especificación	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 requisitos mínimos. • Procesador 2.0 GHz • Memoria 512 MB. • Disco duro 20 GB. • Sistema Operativo Windows 98, 2000, XP o para Servidor o Linux. • Navegador internet Explorer 6.0 o posterior, Mozilla Firefox 2.X • Conexión a Internet. mínimo 56Kbps Para que un cliente de la aplicación pueda observar y analizar resultados de los procesos consignados en documentos electrónicos, el punto de acceso deberá cumplir con los siguientes requisitos mínimos. • Herramienta Microsoft Office 2003 o superior • Herramienta Acrobat Reader 6.0 o superior • Herramienta Open Office 2.3 o superior
Identificador	RNF_C.25
Tipo	Capacidad

2.3 Actores del Sistema

Los actores del sistema representan los futuros usuarios que interactuarán con el sistema y en específico con las funcionalidades del módulo en cuestión. Se muestra a continuación los actores

identificados, sus relaciones siendo de generalización/especialización en cada caso y una descripción detallada de los mismos:

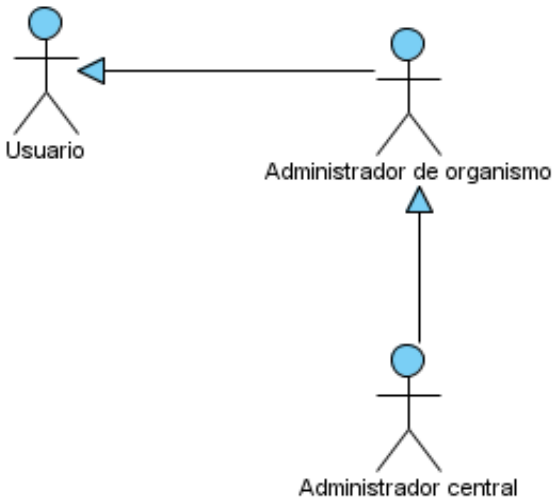


Figura 4: Actores del sistema del módulo Administración.

Tabla 2: Descripción de los Actores del Sistema

Actor	Descripción
Usuario	El actor "Usuario" es un usuario común del sistema por lo que tiene permiso para autenticarse. Es un usuario que puede cambiar su contraseña cuando lo desee y restaurarla cuando la haya perdido u olvidado.
Administrador de organismo	El actor **"**Administrador de organismo" es un usuario del sistema que puede comportarse como un actor "Usuario", por lo que va tener los mismos privilegios que este. Además este usuario puede restaurar la contraseña de cualquier otro usuario de su mismo organismo que la haya perdido y crear usuario, modificar usuario, eliminar usuario y ver los detalles de estos últimos (siempre que estos pertenezcan a su organismo).

Administrador central	El actor **“**Administrador central” es un usuario del sistema que puede comportarse como un actor “Administrador de organismo”, por lo que va tener los mismos privilegios que este. Además este usuario puede crear los roles que tendrá el sistema, modificarlos, eliminarlos y ver sus detalles, de igual manera puede crear los grupos que tendrá el sistema, modificarlos, eliminarlos y ver sus detalles.

2.4 Casos de Uso de Sistema

Luego de obtener los requisitos del sistema se realizan los casos de uso (CU) del sistema que no son más que la forma en que los actores usan el sistema. Generalmente se define un caso de uso por cada requisito funcional y luego aplicando los patrones de casos de uso, se procede a estructurar con más claridad los mismos, agrupando por funcionalidades como agrupar los CU fuertemente relacionados o dividir los CU sobrecargados o complejos. Principalmente se utiliza el patrón CRUD, a continuación se especifica la lista de los casos de uso del sistema para el módulo de Administración del sistema SINAPSIS:

CU.1 Autenticar

Caso de Uso:	Autenticar
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario trata de entrar al sistema, el mismo permite darle entrada al usuario en el sistema en dependencia del grupo de usuarios al que pertenece, los roles asociados y las funcionalidades asignadas al mismo, terminando cuando el usuario tiene acceso a su ambiente de trabajo.
Referencias:	RF_A.1, RF_A.2, RF_A.3, RF_A.4
Prioridad:	Crítico

CU.2 Modificar contraseña

Caso de Uso:	Modificar contraseña
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el Usuario que esta autenticado necesita modificar su contraseña. Consiste en que el Usuario selecciona la opción Modificar contraseña e introduce la contraseña

	anterior, la nueva y su confirmación. El caso de uso termina con la contraseña del Usuario modificada.
Referencias	RF_A.5, RF_A.6, RF_A.2
Prioridad:	Crítico

CU.3 Restaurar Contraseña

Caso de Uso:	Restaurar contraseña
Actores:	Administrador de organismo
Resumen:	El caso de uso se inicia cuando el Administrador de organismo necesita restaurar la contraseña de un usuario. Consiste en que el Administrador de organismo selecciona la opción Restaurar la contraseña para que el usuario que olvidó su contraseña, introduzca una nueva. El caso de uso termina con la contraseña del Usuario actualizada.
Referencias:	RF_A.7, RF_A.8, RF_A.2, CU Buscar usuario (extiende)
Prioridad:	Crítico

CU.4 Gestionar Usuario

Caso de Uso:	Gestionar usuario
Actores:	Administrador de organismo
Resumen:	El caso de uso se inicia cuando el Administrador de organismo necesita gestionar un usuario. Consiste en que el Administrador de organismo selecciona la opción de adicionar un usuario y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un usuario existente ya sea para modificarlo, eliminarlo o ver sus detalles. El caso de uso termina con la creación, actualización o eliminación de un usuario.
Referencias	RF.10, RF.11, RF.12, RF.13, RF.14, RF.15, RF.2, RF.3, CU Buscar usuario (extiende)
Prioridad	Crítico

CU.5 Buscar Usuario

Caso de Uso:	Buscar usuario
Actores:	
Resumen:	El caso de uso se inicia cuando es invocado desde los casos de uso: Restaurar contraseña y Gestionar usuario. Consiste en que el Administrador de organismo selecciona la opción de buscar un usuario e introduce el o los criterios de búsqueda requeridos. El caso

	de uso termina mostrando el usuario buscado.
Referencias	RF.15, RF.2
Prioridad	Crítico

CU.6 Gestionar rol

Caso de Uso:	Gestionar rol
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el administrador central necesita gestionar un rol. Consiste en que el administrador central selecciona la opción de adicionar un nuevo rol y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un rol ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un rol.
Referencias	RF.16, RF.17, RF.18, RF.19, RF.20, RF.2, CU Buscar rol (extiende).
Prioridad	Crítico

CU.7 Buscar rol

Caso de Uso:	Buscar rol
Actores:	
Resumen:	El caso de uso se inicia cuando es invocado desde el caso de uso Gestionar rol. Consiste en que el Administrador central selecciona la opción de buscar un rol e introduce los criterios de búsqueda requeridos. El caso de uso termina mostrando el rol buscado.
Referencias	RF.21, RF.2
Prioridad	Crítico

CU.8 Gestionar grupo

Caso de Uso:	Gestionar grupo
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un grupo. Consiste en que el Administrador central selecciona la opción de adicionar un grupo y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un grupo existente ya sea para modificarlo, eliminarlo o ver sus detalles. El caso de uso termina con la creación, actualización o eliminación de un grupo.
Referencias	RF.22, RF.23, RF.24, RF.25, RF.26, RF.2, CU Buscar grupo (extiende)

Prioridad	Crítico

CU.9 Buscar grupo

Caso de Uso:	Buscar grupo
Actores:	
Resumen:	El caso de uso se inicia cuando es invocado desde el caso de uso Gestionar grupo. Consiste en que el Administrador central selecciona la opción de buscar un grupo e introduce los criterios de búsqueda requeridos. El caso de uso termina mostrando el grupo buscado.
Referencias	RF.27, RF.2
Prioridad	Crítico

2.5 Diagrama de Casos de Uso del Sistema

Una vez definidos los actores y casos de uso del sistema se compone el diagrama de casos de uso del sistema (DCUS), que representa las relaciones entre los CU y los actores del sistema. Para la realización del diagrama se utilizo el patrón Extensión concreta. Finalmente queda elaborado de la siguiente manera:

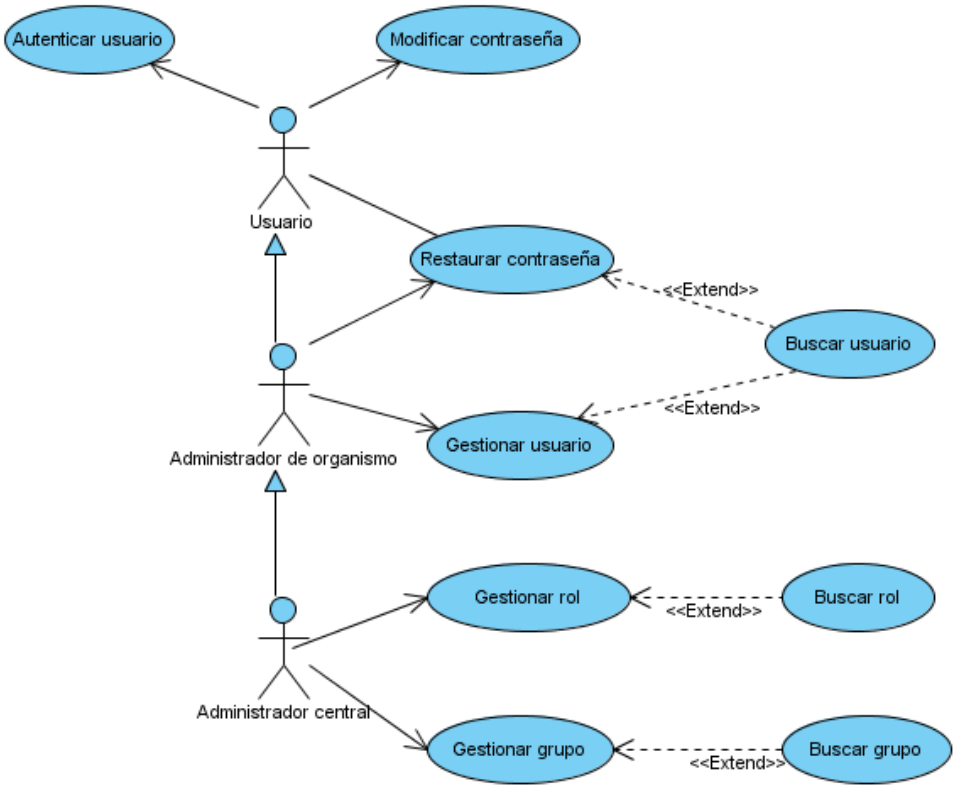


Figura 5: Diagrama de casos de uso del sistema del módulo Administración.

2.6 Descripción de los Casos de Uso del Sistema

Para especificar con detalle cada una de las funcionalidades que deben ser implementadas se describen los 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. A continuación se muestra la descripción de un caso de uso del sistema, para la descripción de todos los casos de uso ver Documento Modelo de Sistema (Proyecto Sinapsis, 2010).

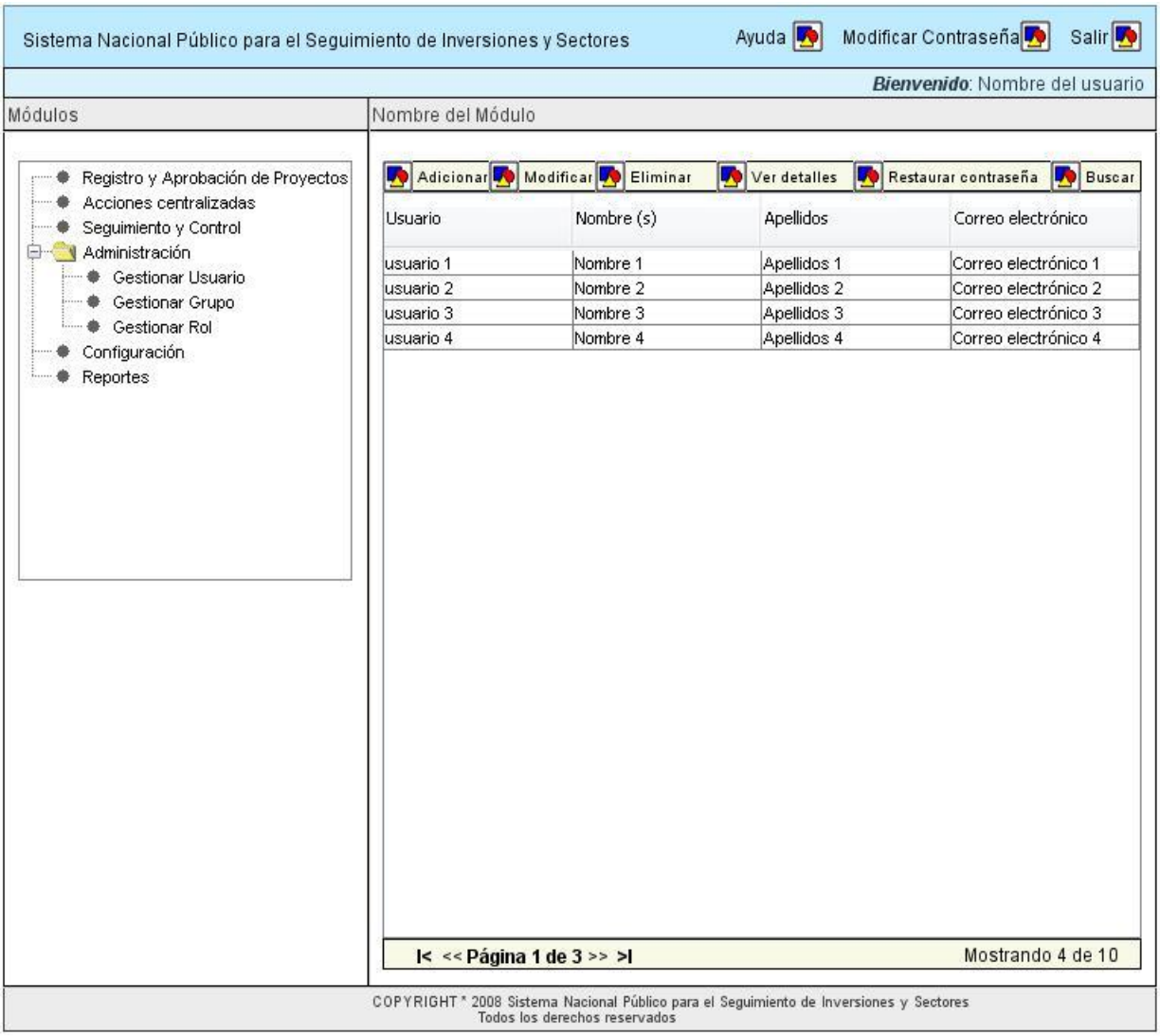
Caso de uso Gestionar usuarios

Caso de Uso:	Gestionar usuario
Actores:	Administrador de organismo
Resumen:	El caso de uso se inicia cuando el Administrador de organismo necesita gestionar un usuario. Consiste en que el Administrador de organismo selecciona la opción de adicionar un usuario y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un usuario existente ya sea para modificarlo, eliminarlo o ver sus detalles. El caso de uso termina con la creación, actualización o eliminación de un usuario.
Precondiciones:	• El sistema debe estar instalado y ejecutándose correctamente. • El usuario debe estar autenticado con los permisos necesarios.
Referencias	RF.10, RF.11, RF.12, RF.13, RF.14, RF.15, RF.2, RF.3, CU Buscar usuario (extiende)
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 Administrador de organismo selecciona la opción “Administración” y dentro de esta la opción “Gestionar usuario”.	2. El sistema muestra una interfaz donde se lista el conjunto de usuarios existentes. De las mismos se muestra: • Usuario • Nombre (s) • Apellidos • Correo electrónico

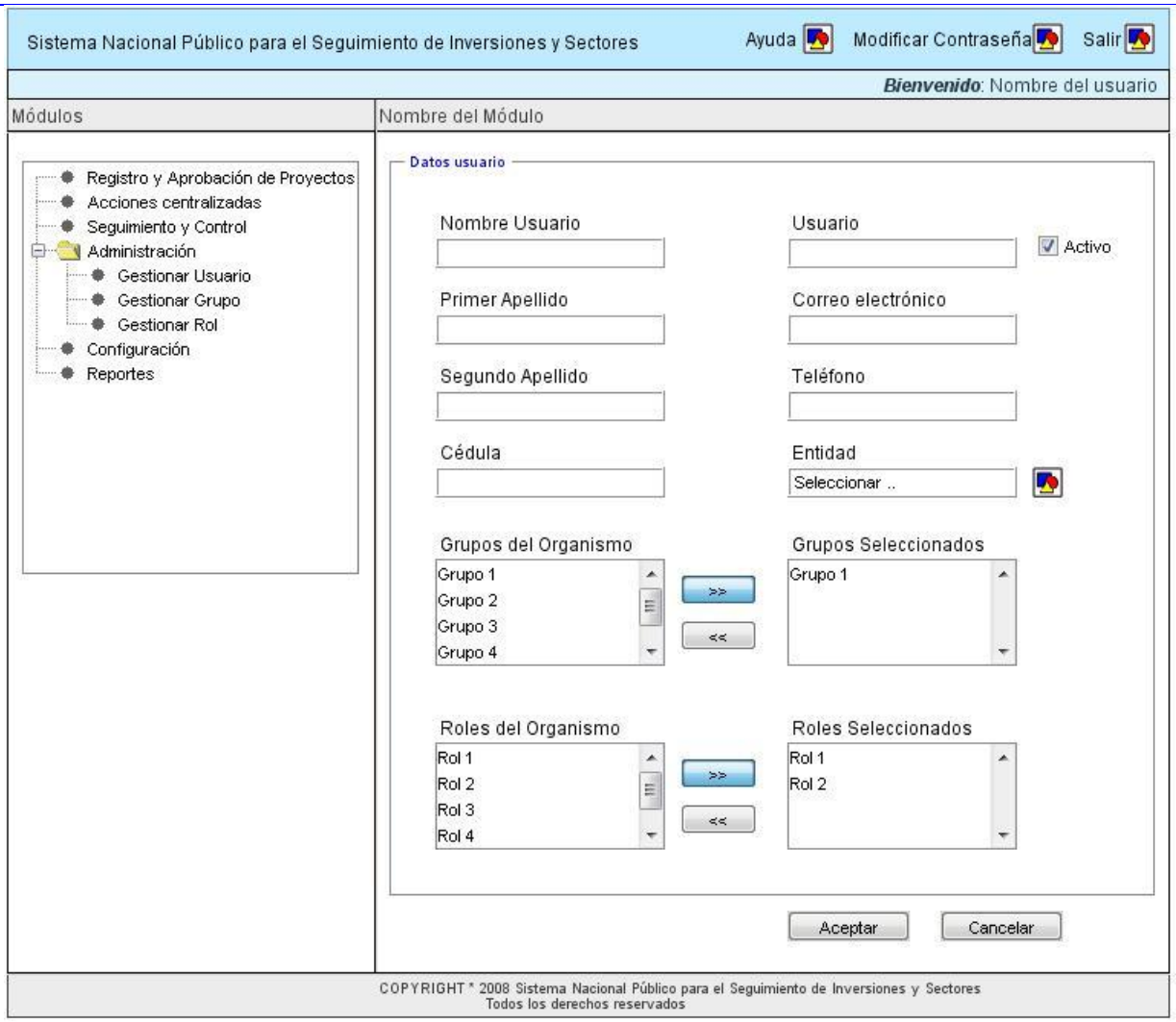
3. El actor Administrador de organismo selecciona una de las siguientes opciones: • Adicionar (Ver sección **"Adicionar usuario"**) • Modificar (una vez que haya seleccionado el usuario que desea modificar, ver sección **"Modificar usuario"**) • Eliminar (una vez que haya seleccionado el usuario, ver sección **"Eliminar usuario"**) • Ver detalles (una vez que haya seleccionado el usuario, ver sección **"Ver detalles usuario"**) • Buscar (se invoca al CU Buscar usuario)	

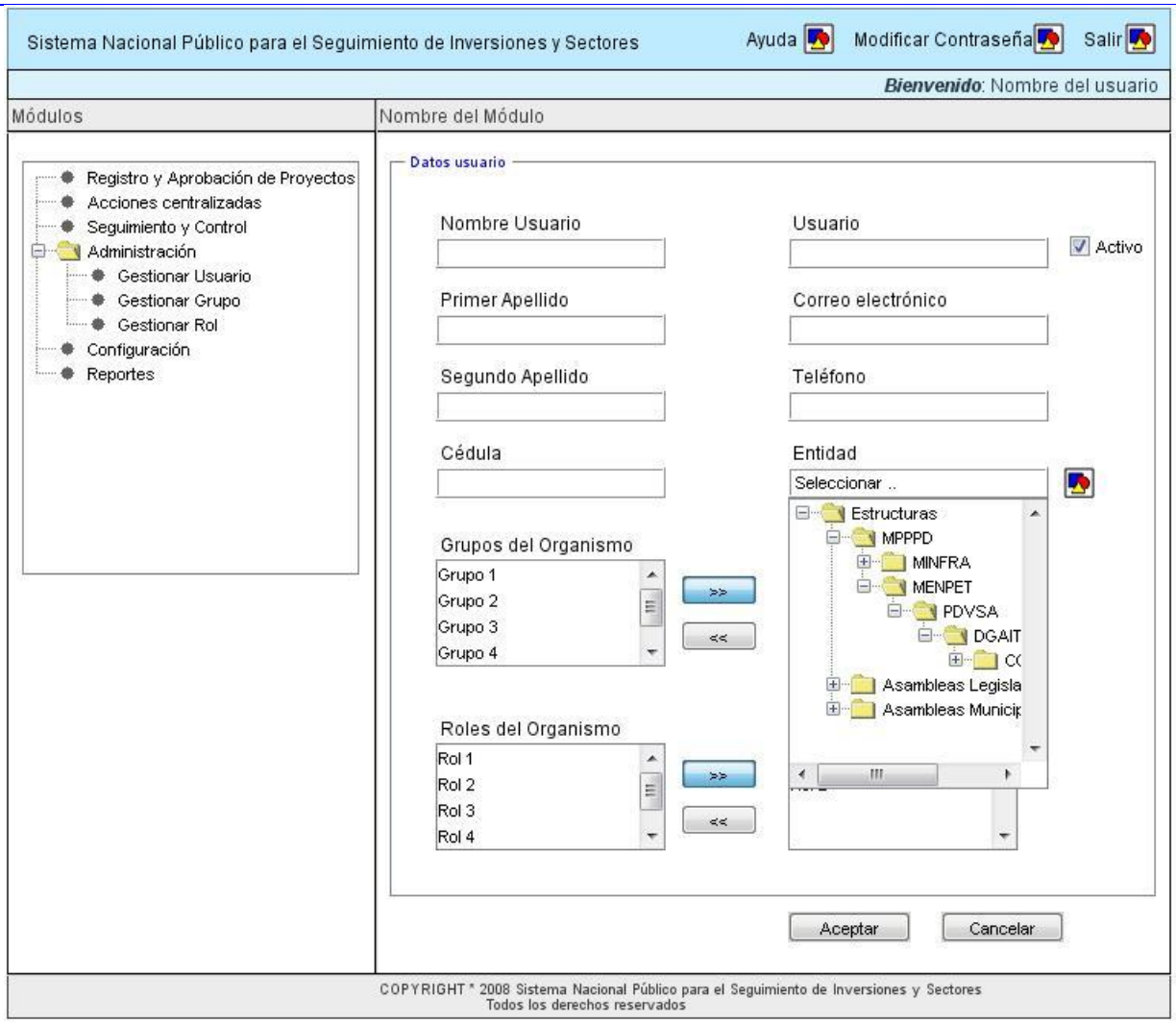
Prototipo de Interfaz



Sección "Adicionar usuario"

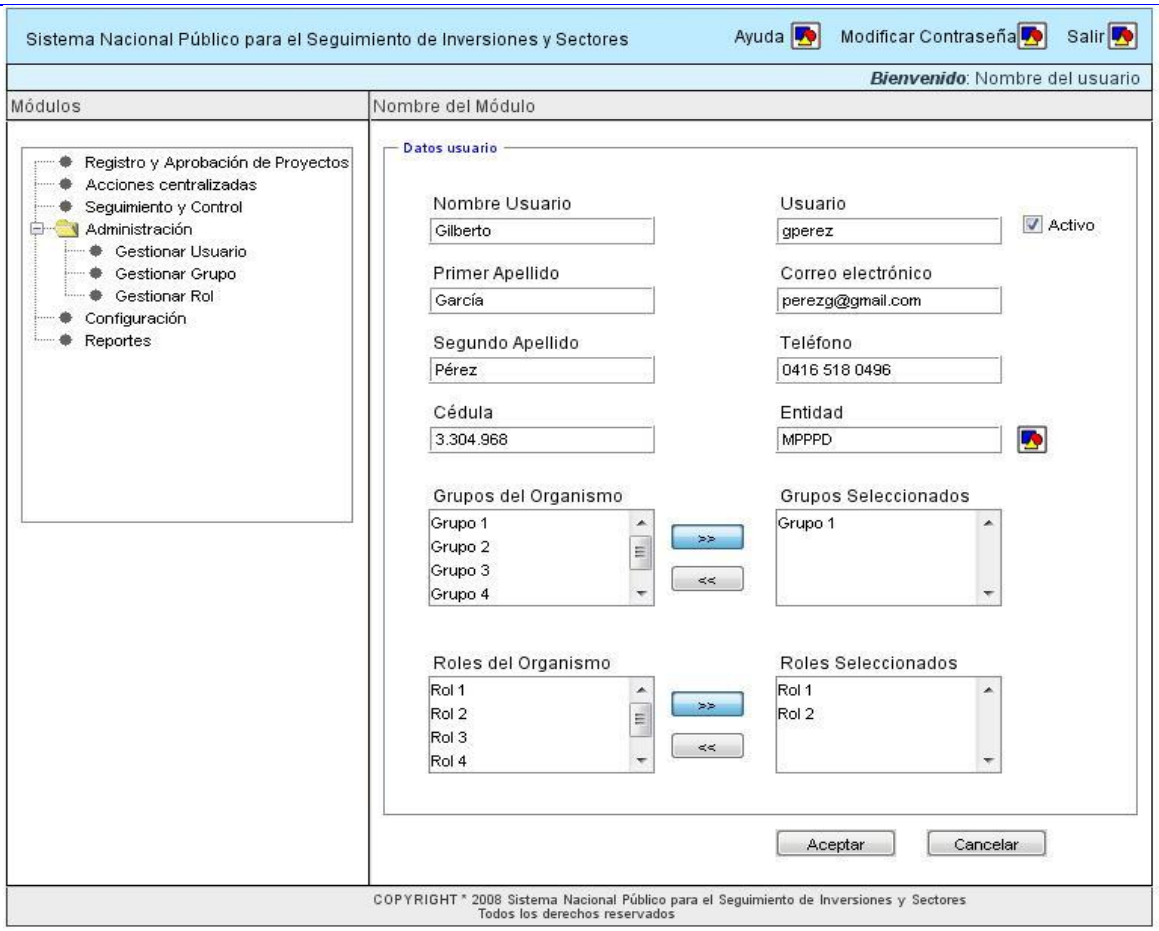
Acción del Actor	Respuesta del Sistema
	1. El sistema muestra la interfaz para crear un usuario, solicitando los siguientes datos: • Nombre persona • Primer apellido • Segundo apellido • Cédula • Usuario • Activo (activado por defecto) • Correo electrónico • Teléfono • Entidad a la que pertenece • Grupos que le serán asignados • Roles que le serán asignados
2. El actor Administrador de organismo introduce los datos correspondientes y selecciona la opción “Aceptar”.	3. El sistema valida que los datos introducidos son correctos y que no hay campos obligatorios vacíos.
	4. El sistema crea el usuario, terminando así el caso de uso.
Prototipo de Interfaz	


Sistema Nacional Público para el Seguimiento de Inversiones y Sectores
Ayuda
Modificar Contraseña
Salir
Bienvenido: Nombre del usuario
Módulos
Nombre del Módulo
Registro y Aprobación de Proyectos
Acciones centralizadas
Seguimiento y Control
Administración
Gestionar Usuario
Gestionar Grupo
Gestionar Rol
Configuración
Reportes
Datos usuario
Nombre Usuario
Usuario
Activo
Primer Apellido
Correo electrónico
Segundo Apellido
Teléfono
Cédula
Entidad
Seleccionar ..
Grupos del Organismo
Grupo 1
Grupo 2
Grupo 3
Grupo 4
>>
<<
Grupos Seleccionados
Grupo 1
Roles del Organismo
Rol 1
Rol 2
Rol 3
Rol 4
>>
<<
Roles Seleccionados
Rol 1
Rol 2
Aceptar
Cancelar
COPYRIGHT * 2008 Sistema Nacional Público para el Seguimiento de Inversiones y Sectores
Todos los derechos reservados



Flujo alterno al paso 2 "Operación cancelada"	
Acción del Actor	**Respuesta del Sistema**
2. a El actor Administrador de organismo selecciona la opción "Cancelar".	2. b El sistema cancela la operación y regresa al paso 2 del flujo normal de eventos. Terminando así el caso de uso.
Flujo alterno al paso 3 "Datos incorrectos y/o campos vacíos"	
Acción del Actor	**Respuesta del Sistema**
	3. a El sistema valida que los datos introducidos no son correctos y/o que hay campos obligatorios vacíos.
	3. 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

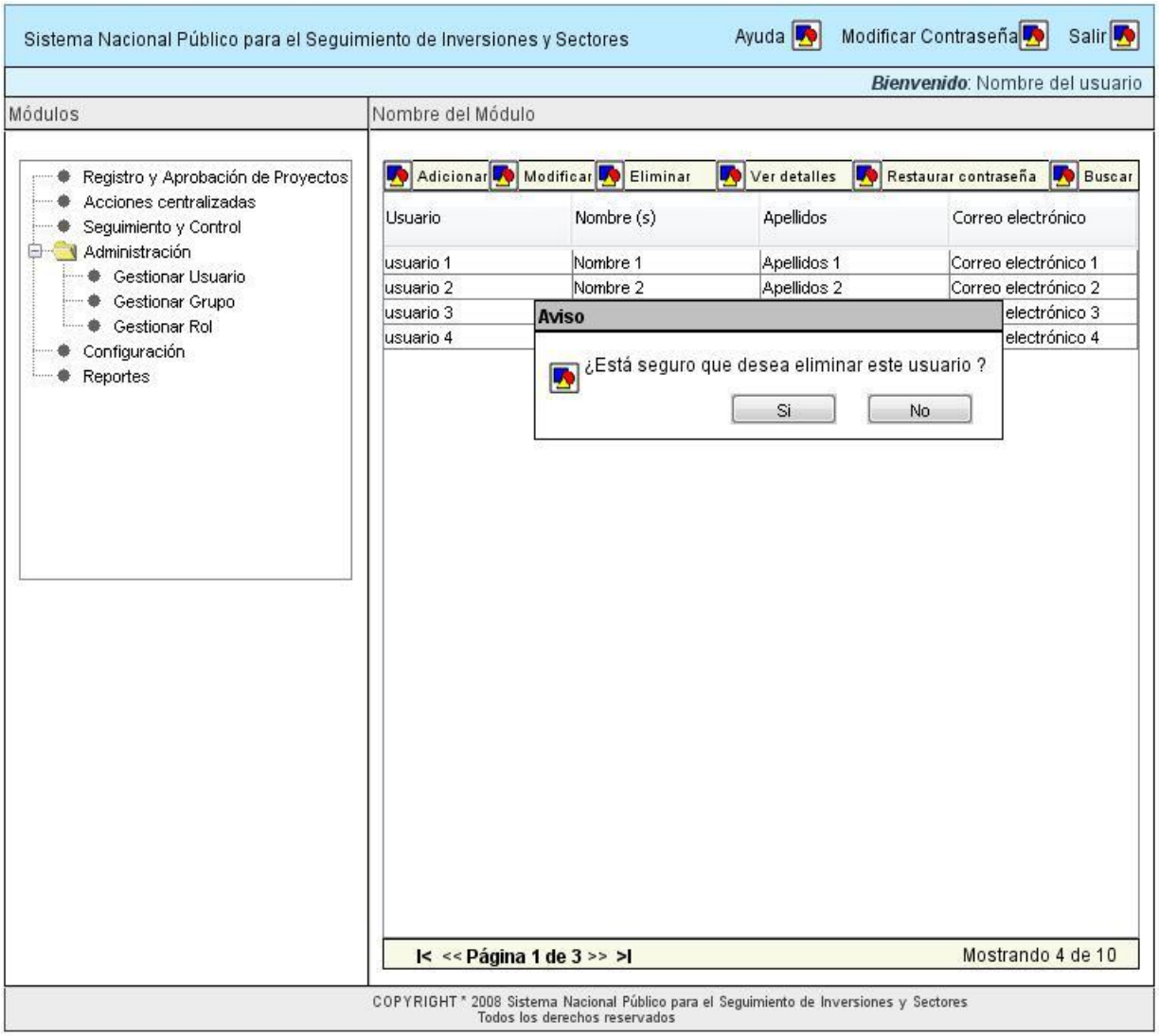
	sección.
Sección “Modificar usuario”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz para modificar los siguientes datos del usuario seleccionado: • Nombre persona • Primer apellido • Segundo apellido • Cédula • Usuario • Activo (activado por defecto) • Correo electrónico • Teléfono • Entidad a la que pertenece • Grupos que le serán asignados • Roles que le serán asignados
2. El actor Administrador de organismo modifica los datos que desea y selecciona la opción “Aceptar”.	3. El sistema valida que los datos introducidos son correctos y que no hay campos obligatorios vacíos.
	4. El sistema actualiza el usuario, terminando así el caso de uso.
Prototipo de Interfaz	



Flujo alterno al paso 2 "Operación cancelada"	
Acción del Actor	**Respuesta del Sistema**
2. a El actor Administrador de organismo selecciona la opción "Cancelar".	2. b El sistema cancela la operación y regresa al paso 2 del flujo normal de eventos.
Flujo alterno al paso 3 "Datos incorrectos y/o campos vacíos"	
Acción del Actor	**Respuesta del Sistema**
	3. a El sistema valida que los datos introducidos no son correctos y/o que hay campos obligatorios vacíos.
	3. 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 "Eliminar usuario"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un mensaje de aviso donde

	pregunta si está seguro que desea eliminar el usuario seleccionado.
2. El actor Administrador de organismo selecciona la opción “Aceptar”.	3. El sistema valida que el usuario puede ser eliminado.
	4. El sistema elimina el usuario, terminando así el caso de uso.

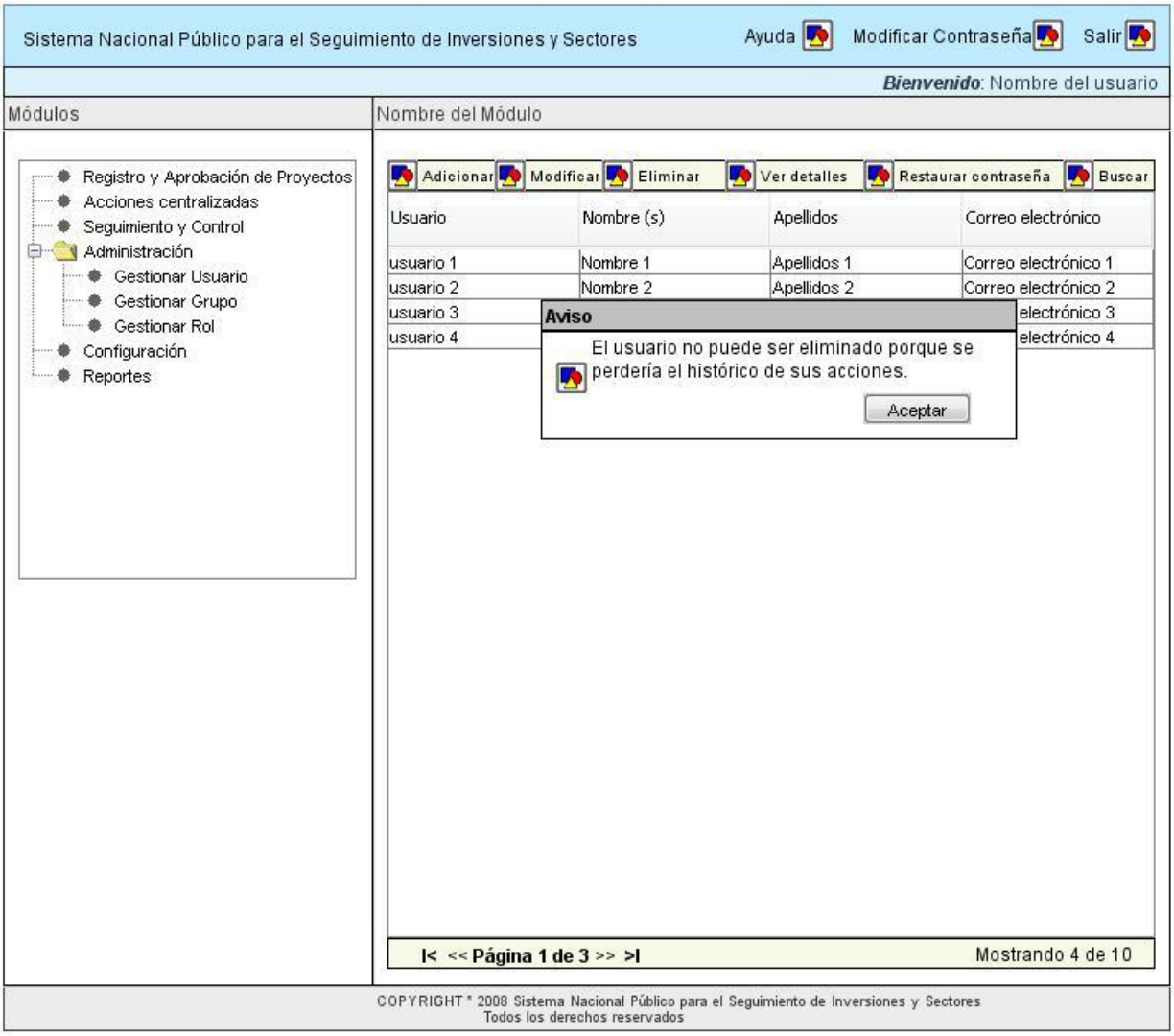
Prototipo de Interfaz



Flujo alterno al paso 2 “Operación cancelada”	
Acción del Actor	**Respuesta del Sistema**
2. a El actor Administrador de organismo selecciona la opción “Cancelar”.	2. b El sistema cancela la operación y regresa al paso 2 del flujo normal de eventos.
Flujo alterno al paso 3 “Usuario con histórico”	
Acción del Actor	**Respuesta del Sistema**

	3. a El sistema valida que el usuario no puede ser eliminado.
	3. b El sistema muestra un mensaje de aviso informando los motivos por los que no puede eliminarse el usuario.
3. c El actor Administrador de organismo selecciona la opción “Aceptar”.	3. d El sistema regresa al paso 2 de del flujo normal de eventos.

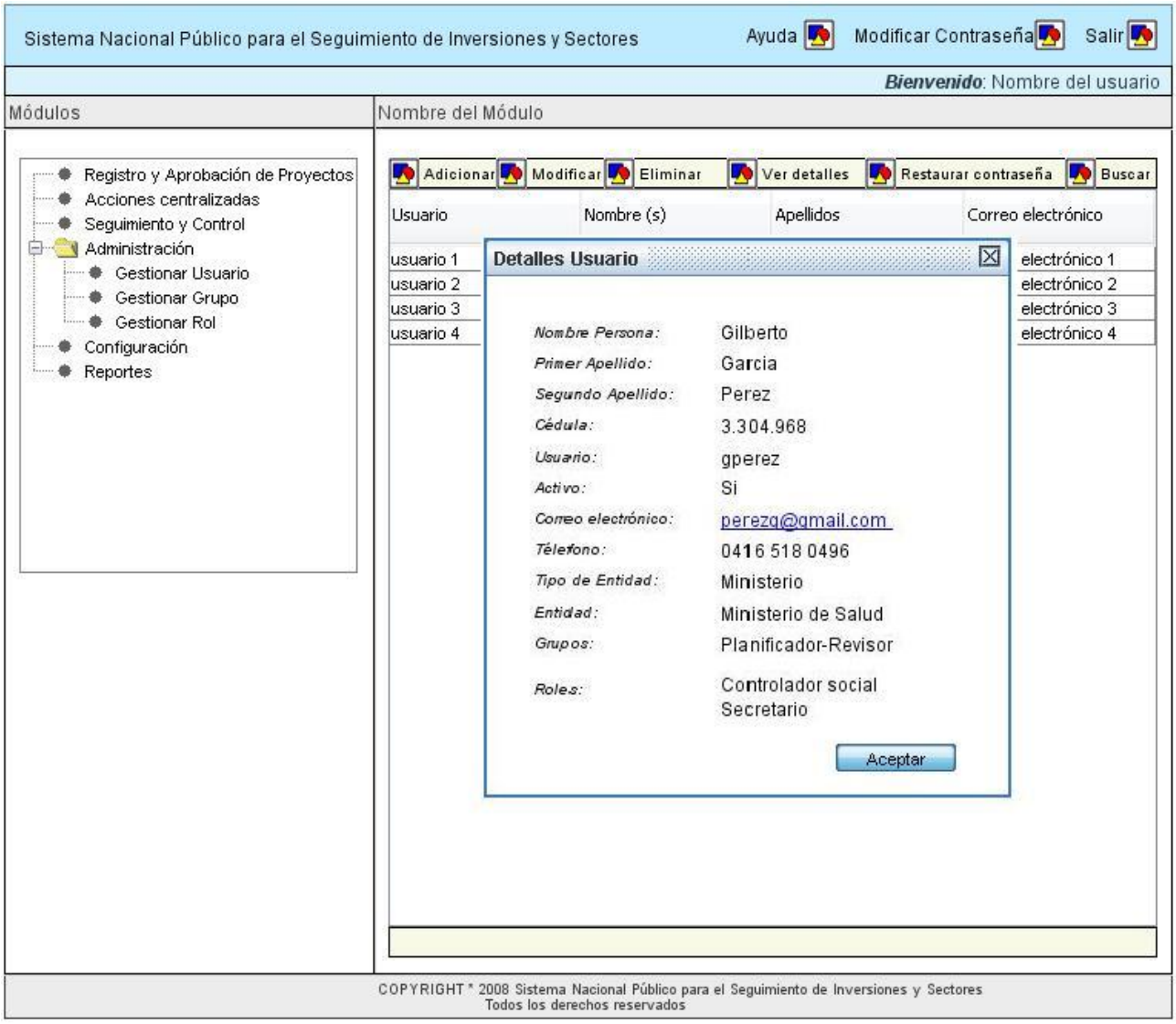
Prototipo de Interfaz



Sección “Ver detalles”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra los siguientes detalles del usuario: • Nombre persona • Primer apellido • Segundo apellido

	• Cédula • Usuario • Activo (activado por defecto) • Correo electrónico • Teléfono • Entidad a la que pertenece • Grupos asignados • Roles asignados

Prototipo de Interfaz



Pos-condiciones	• El sistema queda con un usuario creado. • El sistema queda con un usuario actualizado. • El sistema queda con un usuario eliminado.

2.7 Diseño de software

El diseño del software desempeña un papel importante en el desarrollo de software: permite que la Ingeniería del software produzca los diversos modelos para la solución que se pondrá en desarrollo.

El diseño se define como “el proceso para definir la arquitectura, los componentes, los interfaces, y otras características de un sistema o un componente” y “el resultado de este proceso.” Visto como proceso, el diseño del software es la actividad del ciclo de vida de la cual los requisitos del software se analizan para producir una descripción de la estructura interna del software que servirá como base para su construcción. (IEEE Swebok, 2005).

En los epígrafes posteriores se desarrollarán los distintos artefactos que se generan en el modelo de diseño del módulo Administración del sistema SINAPSIS.

2.8 Subsistema de diseño

Un subsistema de diseño encapsula comportamiento, proporcionando interfaces explícitas y formales, y no expone el contenido interno. Esto proporciona la capacidad de encapsular completamente las interacciones de una serie de clases y/o subsistemas. Su utilización viene dada en el empaquetamiento de elementos del modelo de diseño en estructuras más independientes

Los subsistemas de diseño son un importante medio de descomposición de grandes sistemas en componentes comprensibles. Son particularmente útiles para especificar componentes de los que se espera que se desarrollen de forma independiente, se vuelvan a utilizar o se sustituyan. Se muestran los subsistemas relacionados con el módulo Administración del sistema SINAPSIS:

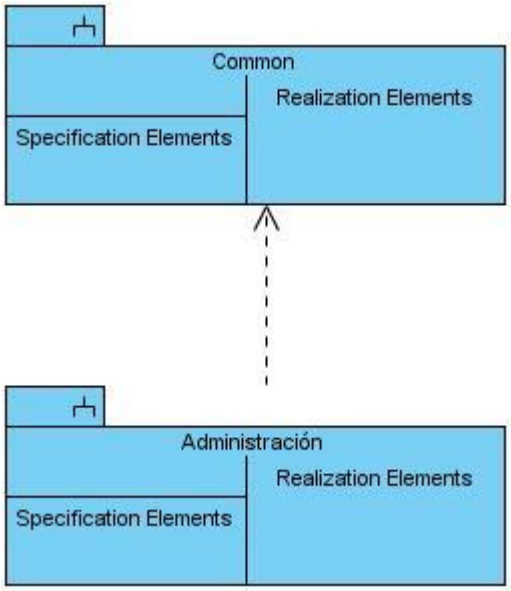


Figura 6: Subsistemas de diseño del Módulo Administración.

2.9 Paquetes de diseño

Los paquetes de diseño son una recopilación de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes. Se utiliza para estructurar el modelo de diseño dividiéndolo en componentes más pequeños en especial para facilitar su comprensión y como herramienta organizativa. A diferencia de los subsistemas de diseño un paquete de diseño no ofrece una interfaz formal, aunque puede exponer algunos de sus contenidos públicos que ofrecen comportamiento. Se definieron los siguientes paquetes para el subsistema de Administración del sistema SINAPSIS:

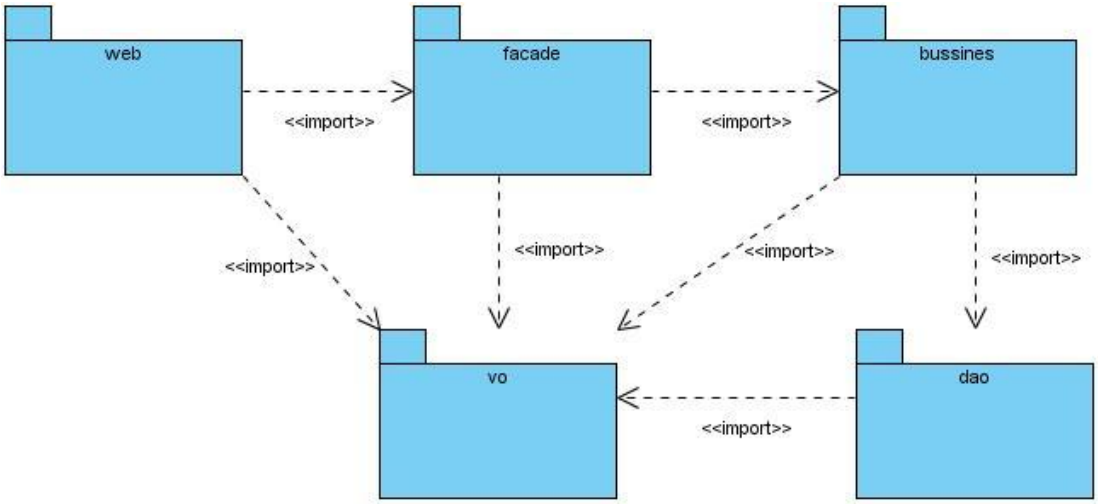


Figura 7: Paquetes de diseño del módulo Administración.

2.10 Diagrama de clases de diseño

Los diagramas de clase incluyen clases y sus relaciones con otras clases para ilustrar la estructura estática de un modelo de objeto. Los diagramas de clases del diseño constituyen la vista del diseño estático de un sistema.

El sistema SINAPSIS por sus fines de uso se definió que debía ser una aplicación web, por tal razón en los diagramas de clases de diseño elaborados se hicieron uso de los estereotipos web, con el objetivo de ilustrar la colaboración real y la representación de todos los elementos que interactúan en la ejecución de las funcionalidades que debe brindar el sistema.

Teniendo en cuenta el lenguaje Java para la implementación del sistema junto con el framework Spring se hicieron uso de los patrones de diseño para el diseño de clases, principalmente el patrón Façade para la comunicación entre las capas y el patrón DAO para crear una interfaz común que garantice el acceso a los datos.

Se muestran a continuación los principales diagramas de clases del diseño del módulo Administración del sistema SINAPSIS:

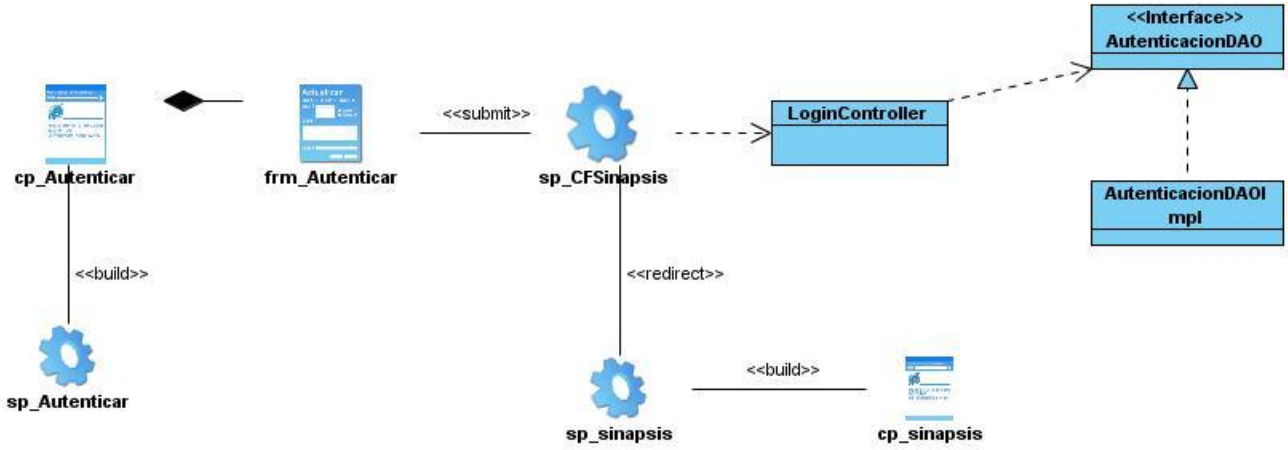


Figura 8: Diagrama de clases del diseño, CU Autenticar usuario.

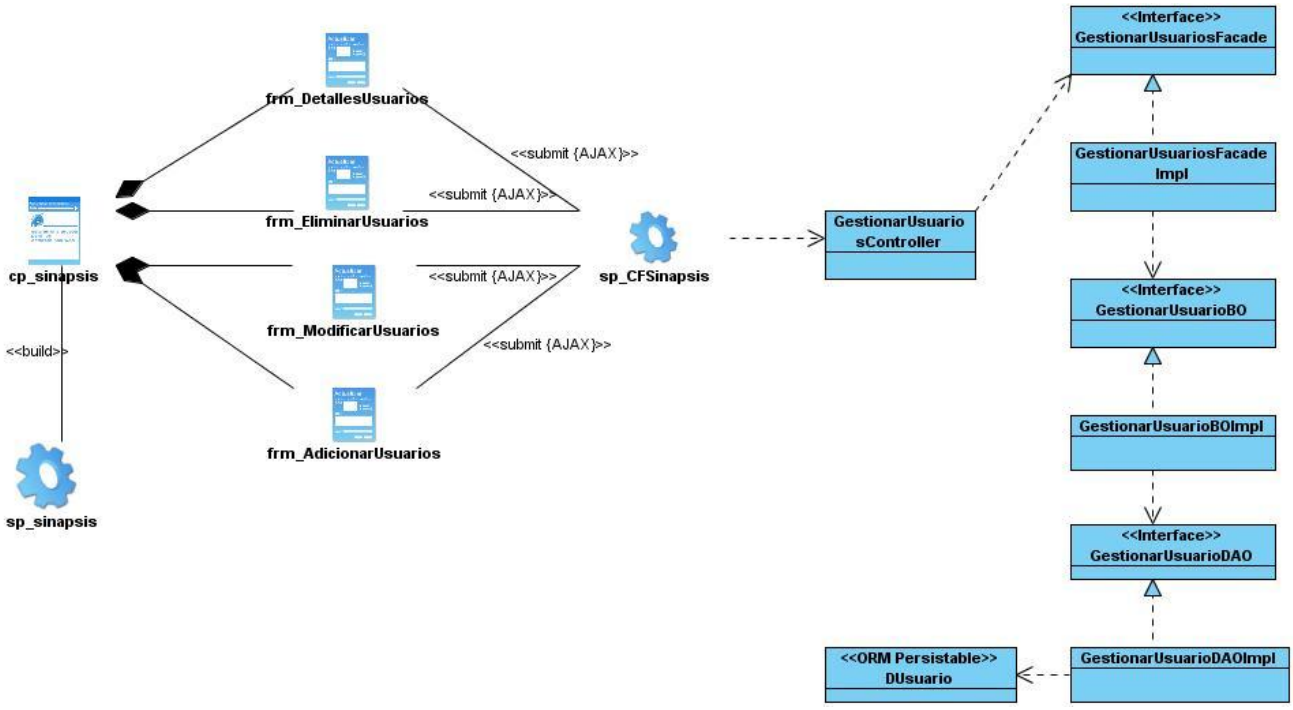


Figura 9: Diagrama de clases del diseño, CU Gestionar Usuarios.

2.11 Diagrama de secuencia

A continuación se muestran los diagramas de secuencia para el caso de uso Autenticar y para el escenario Adicionar del caso de uso Gestionar Usuario. Los restantes diagramas del módulo Administración del sistema SINAPSIS ver Documento Modelo de Sistema.

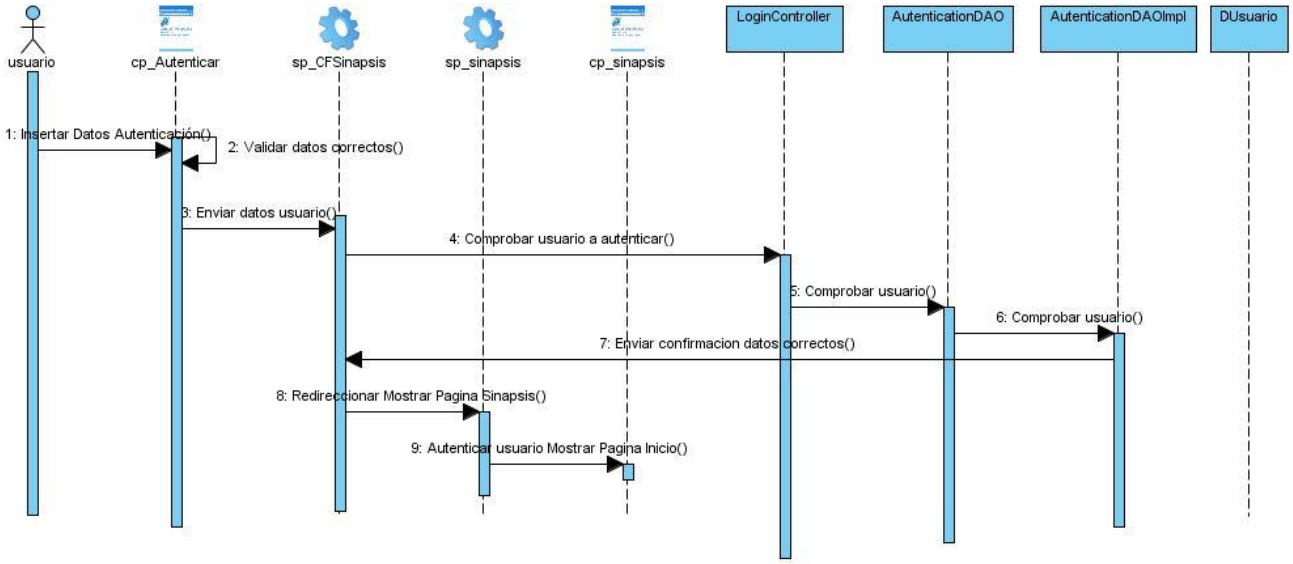


Figura 10: Diagrama de Secuencia del CU Autenticar Usuario.

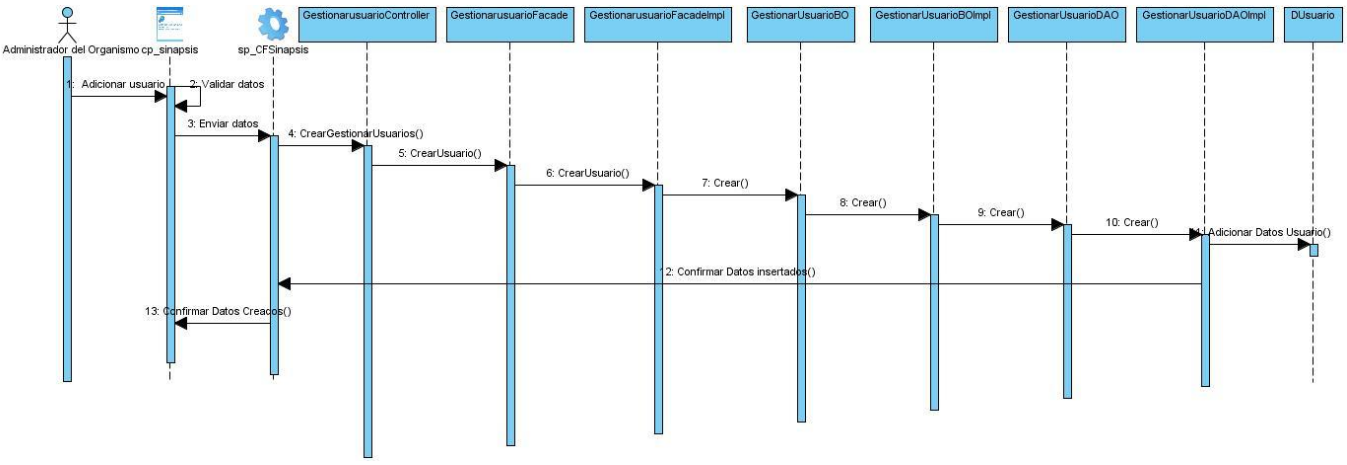


Figura 11: Diagrama de secuencia del escenario Adicionar Usuario, CU Gestionar Usuario.

2.12 Diagrama de clases persistentes

Para facilitar la persistencia de los datos y para poder generar la base de datos y los objetos persistentes junto a sus ficheros de configuración que necesita el framework *Hibernate* para establecer el mapeo de estos objetos con las tablas de la base de datos se generó el diagrama de clases persistentes. Este diagrama representa la relación de los objetos persistentes con sus respectivas relaciones, dependencias y cardinalidades. La sincronización del modelo de entidad relación con este diagrama quedó de la siguiente manera:

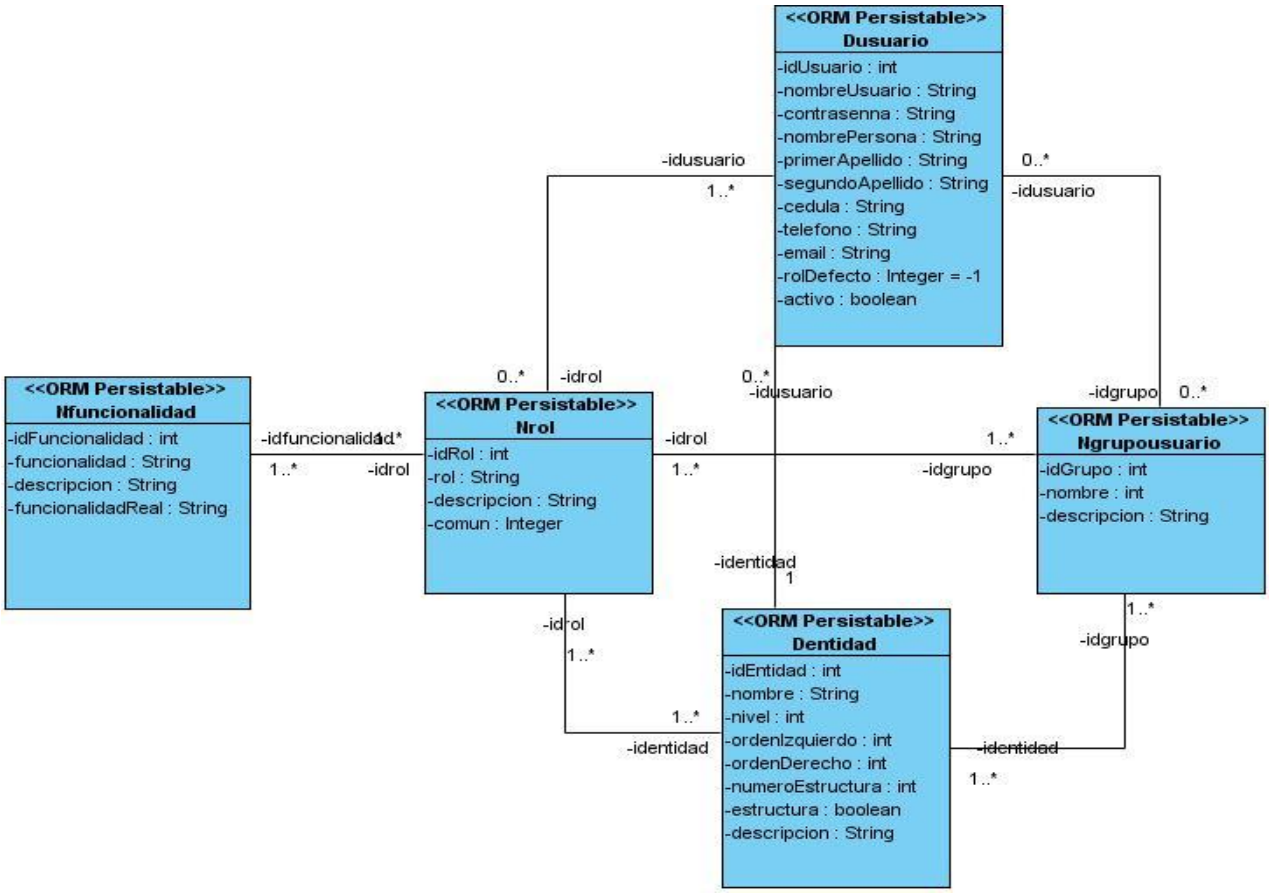


Figura 12: Diagrama de clases persistentes del módulo Administración del sistema SINAPSIS.

2.13 Modelo de datos

El modelo de datos describe el comportamiento lógico y físico de los elementos persistentes de utilidad para dar soporte de información al sistema. Para garantizar la persistencia de los datos se modeló y normalizó el modelo de entidad relación del módulo Administración del sistema SINAPSIS, el cual se muestra a continuación:

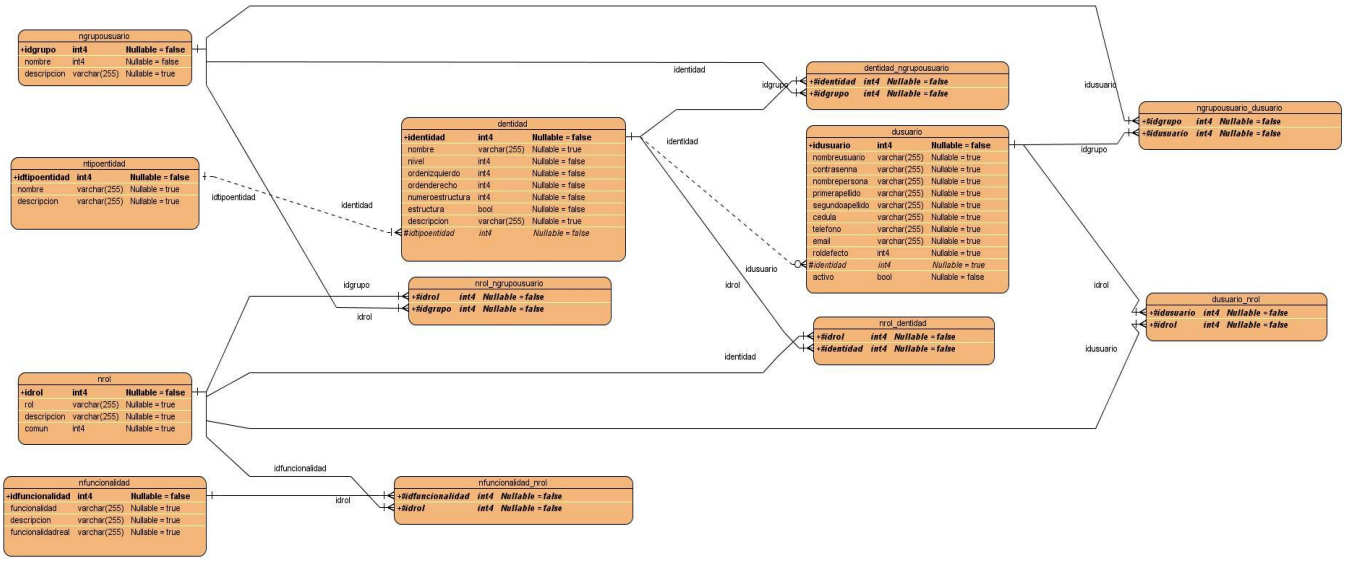


Figura 13: Modelo Entidad Relación del módulo Administración del sistema SINAPSIS.

2.14 Conclusiones

En el capítulo se analizaron y construyeron los elementos imprescindibles que forman parte de los requisitos y diseño del módulo de Administración del sistema SINAPSIS, llegando a la siguiente conclusión:

- La aplicación de técnicas de elicitación y un sólido proceso de Ingeniería de requisitos propició una correcta obtención de los mismos, aumentando su comprensión para conformar lo que realmente debe cumplir el sistema.
- La identificación y especificación de los artefactos del modelo del sistema facilitó un mayor entendimiento y un acuerdo común entre los clientes y los desarrolladores, en cuanto a la concepción de las funcionalidades que el sistema debe cumplir.
- La modelación de los artefactos, tanto de los elementos del sistema como del diseño, utilizando Visual Paradigm, facilitó un mayor entendimiento entre los involucrados en el módulo de Administración del sistema SINAPSIS.
- La construcción de los artefactos del modelo de diseño permitió obtener cada uno de los elementos que soportan los requisitos funcionales y no funcionales del sistema, siendo estos la entrada a las actividades de implementación.
- La aplicación de patrones, tanto de casos de uso como de diseño propiciaron obtener artefactos aceptables, más flexibles y reutilizables.

Capítulo 3. Validación.

3.1 Introducción

En el presente capítulo se muestran los métodos empleados para validar los artefactos obtenidos. En el MCUS se aplican las listas de chequeos y las métricas para la calidad de la especificación de requisitos y la funcionalidad del diagrama de casos de uso, así como las actas de liberación llevadas a cabo por el equipo de calidad en la facultad. Para la validación del modelo de diseño se realizaron las métricas a nivel de componentes y tamaño de clase.

3.2 Validación del Modelo de Sistema

Los requisitos identificados para el módulo Administración tanto funcionales como no funcionales se especificaron en el artefacto Documento de Especificación de Requisitos Funcionales y Especificación de Requisitos No Funcionales respectivamente, que junto con el Documento de Modelo de Sistema que recoge la identificación y descripción de casos de uso y actores constituyen los principales artefactos producto del análisis de los requisitos.

Con el objetivo de garantizar su factibilidad, fueron sometidos a varias iteraciones de revisiones por parte de calidad a nivel de proyecto y facultad. Luego de aplicárseles listas de chequeo para la especificación de requisitos (Ver Anexo.1) y para el documento Modelo de Sistema (Ver Anexo.2), e identificadas y corregidas las No Conformidades durante las iteraciones, se liberaron los artefactos anteriormente mencionados del módulo de Administración del sistema SINAPSIS. Ver 0

Posteriormente fueron presentados a los clientes, los cuales procedieron a su revisión con el objetivo de verificar que la especificación mostrada cumplía con todas las expectativas y condiciones del módulo Administración del sistema SINAPSIS, siendo aceptados y quedando legalmente firmadas en el acta de aceptación. Ver Anexo.5

Métrica para la calidad de especificación de requisitos de software

Los requisitos del módulo de Administración fueron comprobados para determinar su especificidad (ausencia de ambigüedad) según la métrica para valorar la calidad de la ERS propuesta por David y sus colegas, basada en la consistencia de la interpretación de los revisores para cada requisito.

Consiste en dividir el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas (n_{ui}) entre la cantidad de requisitos totales (n_r) la suma de los requisitos funcionales y no funcionales:

$$\mathbf{Q_i = n_{ui} / n_r} \qquad \mathbf{n_r = n_f + n_{nf}}$$

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

La relación de los requisitos según las interpretaciones de los miembros del equipo de inspección queda reflejada en la siguiente tabla:

Especificidad			
Tipo de requisito	**Interpretaciones**		**Total**
	Iguales	**Desiguales**	
Funcionales	23	3	26
No funcionales	24	1	25
Total	47	4	51

$\mathbf{n_f} = 26$ $\mathbf{n_{nf}} = 25$ $\mathbf{n_r} = n_f + n_{nf} = 26 + 25 = 51$ $\mathbf{Q_i} = n_{ui} / n_r = 47/51 = 0.921$

El valor resultante de Q1 igual a 0.921, valor muy aproximado a 1 demuestra que el grado de ambigüedad en la especificación de los requisitos de software del módulo Administración fue muy bajo y por consiguiente se comprobó calidad en la especificación.

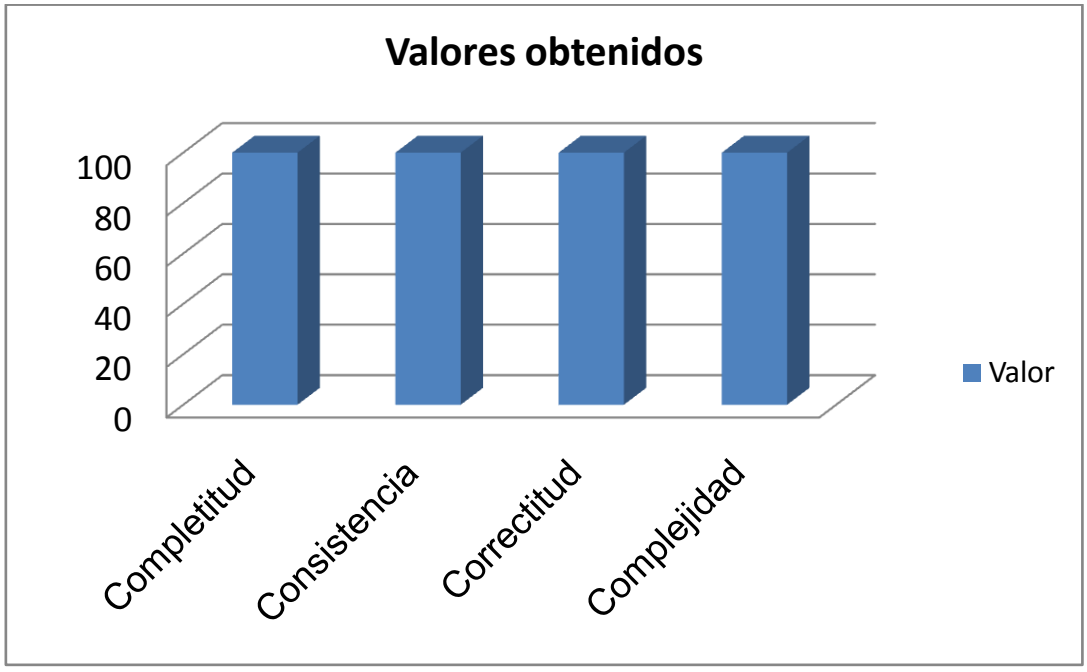
Métrica para la funcionalidad del diagrama de casos de uso del sistema.

Las métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema definen cuatro atributos:

- Completitud: permite determinar el grado en que se ha incluido de forma clara y concisa todos los elementos necesarios para la descripción.
- Consistencia: permite definir el grado en que los elementos del DCUS representan en forma única y no contradictoria un aspecto del problema.
- Correctitud: permite establecer el grado de adecuación del DCUS para satisfacer los requisitos.
- Complejidad: permite medir el grado de claridad y re-uso del DCUS.

Los atributos son analizados por un conjunto de factores en varias iteraciones, cada uno de los cuales tiene asociada una o varias métricas. Estas tomarán un valor en porciento, se muestran en el Anexo.3.

Los resultados obtenidos muestran un valor arrojado por todas las métricas lo que demuestra que se construyó un diagrama de casos de uso del sistema con 100% de calidad de su funcionalidad, como se refleja en la siguiente gráfica:



3.3 Validación del Modelo de Diseño

Como artefacto principal del diseño se obtiene el Documento Modelo de Diseño. Siendo revisado por calidad a nivel de proyecto y facultad y después de corregir las No conformidades en varias iteraciones, fue liberado junto con los artefactos del sistema, entregándose de esta forma el acta de certificado de calidad. Ver Anexo.4

Métricas de diseño arquitectónico

Para comprobar la complejidad del sistema se calcula por la suma de complejidades estructurales y de datos. Ver epígrafe 1.8. En cada caso es necesario conocer:

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

$V(i)$: Número de variables de entrada y salidas del módulo Administración.

La siguiente tabla recoge los resultados de la aplicación de la métrica:

Complejidad		
Estructural	**Datos**	**Sistema**
$S(i) = f_{out}^2(i)$	$D(i) = \frac{V(i)}{f_{out}(i) + 1}$	$C(i) = S(i) + D(i)$

$S(i) = 1^2$	$D(i) = \frac{10}{1+1}$	$C(i) = 1 + 5$
$S(i) = 1$	$D(i) = 5$	$C(i) = 6$

Según la complejidad del sistema se proponen tres tipos: No muy complejo, Complejo y Muy complejo. El umbral para los sistemas no muy complejos requiere cumplir con $D(i) <= 5$ y $S(i) <= 32$, siendo de este tipo nuestro sistema.

Métricas orientadas a clases

Tamaño de Clase

Se utilizó la métrica de tamaño de clase con el objetivo de propiciar un adecuado diseño y comprobar el nivel de reutilización en las clases. Ver epígrafe 1.8.

A continuación se muestran las clases del módulo en las que se aplicará la métrica:

Clases	N°. de atributos	N°. de operaciones
1.GestionarUsuarioFacadeImpl	1	10
2.GestionarRolFacadeImpl	1	7
3.GestionarGrupoFacadeImpl	1	8
4.CambiarContrasennaFacadeImpl	1	1
5. GestionarUsuarioBOImpl	0	10
6. GestionarRolBOImpl	0	8
7. GestionarGrupoBOImpl	0	7
8. CambiarContrasennaBOImpl	0	2
9. GestionarUsuarioDAOImpl	0	11
10.GestionarRolDAOImpl	0	6
11.GestionarGrupoDAOImpl	0	7
12.CambiarContrasennaDAOImpl	0	2
13.GestionarUsuarioController	0	7
14.GestionarRolController	0	5
15.GestionarGrupoController	0	5
16.CambiarContrasennaController	0	1
17.UsuarioController	0	2
18.LoginController	0	1
19.LogOutController	0	1

Se grafican los resultados según la cantidad de clases por categorías (Ver 0 Epígrafe 1.8):

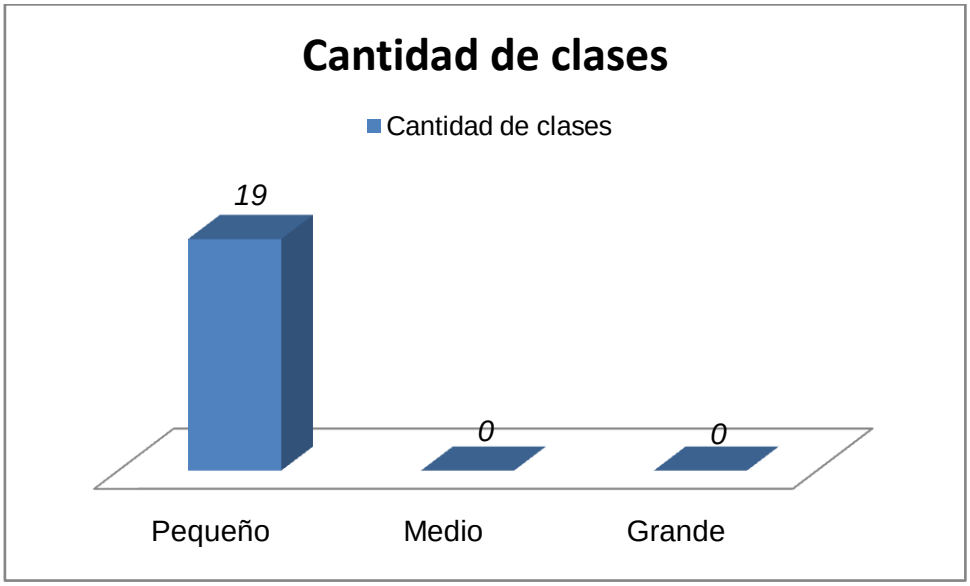


Figura 14: Número de clases por categorías

Finalmente se presentó un total de 19 clases para un promedio de atributos de 0.21 y un promedio de operaciones de 5.31. De esta forma se concluye que tamaño general de clases se encuentra en el umbral de clases pequeñas, indicando una responsabilidad mínima de las clases favoreciendo la reutilización necesaria entre las mismas y facilitando el proceso de construcción del módulo.

3.4 Conclusiones

Al analizar los resultados obtenidos se puede concluir lo siguiente:

- Se comprobó la ausencia de ambigüedad con la aplicación de las métricas para la calidad de la especificación de requisitos.
- La aplicación de métricas para la funcionalidad del DCUS demostró una buena calidad en la construcción del mismo.
- El acta de aceptación por parte de los clientes demuestra que la especificación de requisitos y el MCUS cubren sus expectativas y necesidades.
- Los valores de 1 y 5 para la complejidad estructural y de datos respectivamente indican que el sistema no es muy complejo.
- El 100 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del módulo Administración.
- Quedó demostrada una buena definición y especificación de los artefactos del MCUS y modelo de diseño con las actas de liberación de calidad a nivel de la facultad.

Conclusiones Generales

Al concluir el trabajo se arriba a lo siguiente:

- La aplicación de técnicas para la identificación requisitos propició que se obtuvieran resultados satisfactorios en la elaboración de los requisitos que debe cumplir el sistema.
- La confección de los artefactos: Modelo de Sistema y Especificación de requisitos de software facilitó un entendimiento común entre los desarrolladores y clientes.
- La elaboración de los elementos del modelo de diseño facilita la posterior construcción del sistema que cumpla con los requisitos del software aprobados por el cliente.
- La aplicación de métricas validaron los artefactos en función de su calidad obteniéndose resultados positivos.

Recomendaciones

A lo largo del desarrollo de este trabajo surgieron ideas que se recomiendan con vista al desarrollo del módulo Administración:

1. Seguir utilizando RUP como metodología de desarrollo de software para generar los artefactos correspondientes a los siguientes flujos de trabajo definidos por este proceso.
2. Seguir utilizando Visual Paradigm como herramienta CASE para la construcción de los artefactos restantes del proceso de desarrollo.
3. Desarrollar la implementación del módulo Administración del sistema SINAPSIS.
4. Aplicar ingeniería inversa de código para mantener actualizados los diagramas de clases del diseño.
5. Realizar la administración de requisitos de software para su posterior seguimiento aplicando la Ingeniería de requisitos.

Bibliografía

Armas, M. (2009). *Extreme Programming.* Obtenido de Lider de Proyecto: http://www.liderdeproyecto.com/articulos/extreme_programming.html

Christopher, A. (1979). *The Timeless Way of Building.*

Corp, Microsoft. (23 de 5 de 2006). *Información general del producto Microsoft Office Visio 2007.* Obtenido de http://www.microsoft.com/latam/office/preview/programs/visio/overview.mspx.

Corporation, R. S. (2003). Rational Unified Process Extended Help.

Corral, R. (15 de 1 de 2007). Obtenido de ¿Que metodología de desarrollo elegir?: http://geeks.ms/blogs/rcorral/archive/2007/01/15/iquest-que-metodolog-iacute-a-de-desarrollo-elegir.aspx

Durán Toro, A. y. (2000). *Metodología para la Elicitación de Requisitos de Sistemas Software.* Sevilla.

Escribano, G. F. (9 de 12 de 2002). Introducción a Extreme Programming, Ingeniería del Software II.

Gamma y varios, H. J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley.

Garzón, D. J. *Ingeniería de Requerimientos.*

GSInnova . (2007). *Rational Rose Enterprise* . Obtenido de Grupo de soluciones GSInnova: http://www.rational.com.ar/herramientas/roseenterprise.html

IBM. (s.f.). Recuperado el 2009, de IBM Rational Software: http://www-01.ibm.com/software/rational/

IEEE Standard Association. (2004). *IEEE Standard Glossary of Software Engineering Terminology12-1990.* Obtenido de IEEE Standard Association: http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html

IEEE Swebok. (2005). *Guide to the Software Engineering Body of Knowledge.*

Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *El proceso Unificado de desarollo de software.*

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

Microsoft Office Visio. (5 de 4 de 2009). Obtenido de Microsoft Office Visio Homepage: http://office.microsoft.com/es-es/visio/FX100487863082.aspx.

Microsoft TechNet. (30 de 11 de 2006). *http://technet.microsoft.com/es-es/library/bb490157.aspx.*

Övergaard & Palmkvist, G. K. (2004). *Use Cases Patterns and Blueprints.* Addison Wesley Professional.

Penadés, P. L. (2009). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.*

Pressman, R. S. (2005). *Ingeniería de Software, un enfoque práctico. .*

Proyecto Sinapsis. (2010). *Documento de Especificación de Requisitos Funcionales.* Obtenido de http://10.7.12.247/repo/svn

Proyecto Sinapsis. (2010). *Documento de Especificación de Requisitos No Funcionales.* Obtenido de http://10.7.12.247/repo/svn

Proyecto Sinapsis. (2010). *Documento Modelo de Sistema.* Obtenido de http://10.7.12.247/repo/svn

Pyme Actual. (2006). *Calidad del desarrollo Web.* Obtenido de Pyme Actual: http://www.pymeactual.com/desarrollo-web/calidad-desarrollo-web.php

Sanchez, M. A. (7 de 6 de 2004). *Metodologías De Desarrollo De Software.* Obtenido de Informatizate: http://www.informatizate.net

Sommerville, I. (2005). *Ingeniería del Software.* México DF. Editora Pearson, 4(7), 114-156.

Sparx Systems. (2010). *Enterprise Architect.* Obtenido de Sparx Systems: http://www.sparxsystems.com/products/ea/

UDELAB. (2009). *UDELAB*. Recuperado el 2009, de Universidad de las americas Puebla: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo4.pdf

Visual Paradigm. (22 de 2 de 2010). *UML tool for software application development.* Obtenido de Visual Paradigm: http://www.visual-paradigm.com/product/vpuml/

Zhao, D. J. (15 de 06 de 2005). *Comparación de Herramientas de modelado UML: Enterprise Architect y Rational Rose.* Obtenido de ApexNet: http://www.apexnet.com.ar/index.php/news/main/38/event=view

Anexos

Anexo.1 : Listas de chequeo para la especificación de requisitos

Evaluación
Claridad
¿Los requisitos se escriben en lengua comprensible para el usuario/cliente?
¿Hay requisitos que tienen más de una interpretación?
¿Cada requisito característico del producto final se describe con una terminología única?
¿Hay un glosario en el cual los requisitos significativos y específicos están en términos definidos y claros?
¿Se entienden los requisitos para ponerlos en ejecución por un grupo independiente?
Completo
¿El requisito tiene un contenido específico?
Están especificados los cambios posibles a los requisitos.
La probabilidad de cambios está especificada para cada requisito.
¿Se definen todos los términos en cada uno de los requisitos determinados?
¿Tiene el artefacto un índice bien defino?
¿Existen áreas donde está incompleta la información debido a que el desarrollo aún no ha comenzado a especificarse?
¿La información que falta se define en el requisito?
¿Algún requisito necesita una especificación detallada?
¿Algún requisito necesita ser menos especificado?
¿Todos los requisitos se describen ellos mismos?
¿Están incluidos todos los requisitos relacionados con la funcionalidad?
¿Hay requisitos que produzcan inquietud?
¿Están incluidos todos los requisitos relacionados con el funcionamiento?
¿Están incluidos todos los requisitos relacionados con sus cualidades?
¿Están incluidos todos los requisitos relacionados con las interfaces externas?
¿Están incluidos todos los requisitos relacionados con las bases de datos?
¿Están incluidos todos los requisitos relacionados con el software?
¿Están incluidos todos los requisitos relacionados con el hardware? (No todavía)

¿Están incluidos todos los requisitos relacionados con las entradas de datos?
¿Están incluidos todos los requisitos relacionados con las salidas datos?
¿Están incluidos todos los requisitos relacionados con los mensajes de error?
¿Están incluidos todos los requisitos relacionados con la seguridad? (No todavía)
¿Están incluidos todos los requisitos relacionados con la capacidad de mantenimiento del software? (No todavía)
¿Están incluidos todos los requisitos relacionados con la instalación? (No todavía)
Consistencia (Duda)
¿Hay requisitos que describen el mismo objeto que estén en conflicto con otros requisitos con respecto a la terminología?
¿Hay requisitos que describen al mismo objeto que estén en conflicto con respecto a las características?
¿Hay requisitos que describan dos o más acciones que estén en conflicto temporalmente?
Rastreabilidad
¿Los requisitos fueron rastreados en todos los módulos?
Comprobable
¿Existen requisitos que sean imposibles de cumplir?
Modificable
¿En el documento los requisitos se organizan de manera clara y lógica?
¿La redundancia es mínima y sólo se debe a distintos niveles de abstracción o detalle?
Contenido General
¿Cada requisito es relevante al problema y a su solución?
¿El artefacto cumple con la plantilla especificada por calidad interna?
Prototipo de Interfaz
¿Se especifican todas las interfaces utilizadas?
¿Se especifican todas las interfaces del hardware?
¿Se especifican todas las interfaces del software?
¿Se especifican todas las interfaces de comunicaciones?
¿Se especifican todos los requisitos del diseño de interfaz?
Hardware
¿Se especifica el tipo de hardware que necesita la aplicación?
Software

¿Se especifica el software requerido y el sistema operativo?
Trazabilidad
Cada requisito obedece a una necesidad específica de usuario.
Cada requisito tiene su origen en una fuente (documento o persona) específica.
Cada requisito se puede rastrear hacia delante su incorporación en determinados módulos.
Verificabilidad
Cada requisito se puede implementar.
Para cada requisito existe un procedimiento que, ejecutado por una persona o máquina, permite verificar si se cumple
Hay algún requisito que se va a expresar en términos verificables más adelante.

Anexo.2 : Listas de chequeo para la plantilla Modelo de Sistema.

Evaluación
Plantilla Modelo de Sistema
¿Está debidamente identificada la plantilla?
¿La plantilla cuenta con una sección de ***Control de Versiones***?
¿La plantilla cuenta con un ***Índice General***, con una jerarquía de tópicos bien definida?
¿La plantilla cuenta con una sección de ***Introducción***?
¿Está definido el ***Propósito*** de la modelación dentro de la introducción?
¿Está definido el ***Alcance*** de la modelación dentro de la introducción?
¿Está definida una sub-sección de ***Definiciones, Acrónimos y Abreviaturas*** dentro de la introducción?
¿Existe una sección destinada a recoger las ***Referencias*** dentro de la introducción?
¿La plantilla cuenta con una sección donde se identifiquen y se describan los ***Actores del Sistema***?
¿La plantilla cuenta con una sección que recoja el diagrama de Casos de Uso del Sistema?
¿La plantilla cuenta con una sección de ***Especificación de Casos de Uso?***
¿El formato de letra utilizado para la plantilla en su totalidad es uniforme y solo diferencia tópicos y conceptos que verdaderamente necesiten ser diferenciados del resto del texto?
¿Es la documentación o redacción formal lo suficiente clara para ser entendida por personas fuera del equipo de proyecto?
¿Las descripciones se escriben de forma comprensible al usuario/cliente?
¿Cada descripción textual está conformada por una terminología única?
¿Hay un glosario de términos específicos y claros que facilite el entendimiento?
¿Existen errores gramaticales?
¿Los párrafos están separados por dos espacios y justificados?
¿Cada apéndice independiente está separado uno del otro con un espacio y ordenado desde la izquierda?
¿Los apéndices están separados del texto anterior y posterior por tres y dos espacios respectivamente?
¿Existen errores ortográficos?
¿Del contraste contra diversas fuentes no surgen errores u omisiones?
¿La organización es de manera clara y lógica?
¿La organización cumple con un estándar aceptado?
¿La redundancia es mínima y sólo se debe a distintos niveles de abstracción o detalle?
¿Se etiquetan todas las figuras, tablas y diagramas?
¿Cada frase aporta a la especificación?
Actores del Sistema
¿Hay correspondencia entre los actores del sistema y los trabajadores del negocio?

¿Están descritos todos los actores de manera breve, concreta y formal?
¿Cada actor del sistema expresa un rol, y no una persona?
¿Cada actor del sistema está involucrado con al menos un caso de uso?
Diagrama de Casos de Uso del Sistema
¿EL conjunto de casos de uso del Sistema conforman todas las funcionalidades que se quiere describir con ellos?
¿Existen varios casos de uso del sistema con nombres similares?
¿El diagrama aparece bien estructurado?
¿El diagrama provee una visión general de los casos de uso del sistema fácil de comprender?
¿Existen muchas relaciones en el diagrama?
¿El diagrama es tan complejo y largo que debería ser dividido en varios diagramas más pequeños?
¿Existe algún caso de uso que no tenga relación alguna con un actor del cual se pueda prescindir?
¿Sólo son mencionados los actores que interactúan con el caso de uso?
Especificaciones de Casos de Uso
¿Cada especificación de casos de uso está debidamente numerada e identificada?
¿Los casos de uso del sistema están alineados con la estrategia del producto final?
¿Cada caso de uso del sistema sustenta al menos un objetivo de lo que se quiere automatizar?
¿Está claramente especificado el propósito del caso de uso?
¿Está definido el nivel de prioridad entre los casos de uso?
¿El caso de uso ejecuta sólo actividades dentro del sistema?
¿Existen varios casos de uso con nombres similares?
¿Los flujos centrales y alternos están totalmente descritos?
¿Cada caso de uso sustenta un objetivo del sistema?
¿Están claramente especificadas las pre condiciones del caso de uso?
¿Están claramente especificadas las pos condiciones del caso de uso?
¿Cada caso de uso está involucrado con al menos un actor?
¿Los nombres de los casos de uso expresan sin ambigüedad la funcionalidad del caso de uso al que representan?
¿El nombre y la descripción son entendibles incluso para personas fuera del equipo de desarrolladores?
¿Están descritos todos los posibles flujos que pertenecen al caso de uso?
¿Está descrito claramente cuando puede variar el orden de las actividades?
¿Las descripciones de los flujos de trabajo central y alternos están bien estructuradas?
¿Se hace referencia a los requisitos implícitos en el caso de uso?
¿El inicio y fin del flujo de trabajo están claramente descritos?
¿Cada relación extendida (extended) está claramente descrita de forma tal que es obvio cómo y cuándo es insertado el caso de uso?

Anexo.3 : Factores en la métrica para la Funcionalidad del diagrama de casos de uso.

Factores	Métricas asociadas	Valor (%)
Completitud		
1. ¿Han sido involucradas todas las áreas funcionales relevantes a las cuales apoyará el sistema?	1. Número de áreas funcionales relevantes omitidas	0
2. ¿Han sido involucradas todas las áreas funcionales secundarias a las cuales apoyará el sistema?	2. Número de áreas funcionales secundarias omitidas	0
3. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/modificar o consultar información?	3. Número de roles relevantes omitidos	0
4. ¿Están definidos todos los requisitos que justifican la funcionalidad del caso de uso?	4. Número de requisitos omitidos por caso de uso 5. Número de casos de uso que tienen requisitos omitidos	0 0
5. ¿Existen requisitos que no han sido considerados en algún caso de uso?	6. Número de requisitos que no son considerados en ningún caso de uso	0
6. ¿Están todas las acciones del flujo de eventos redactadas en función del responsable?	7. Número de acciones del flujo de eventos que no están redactadas en función del responsable 8. Número de casos de uso que tienen acciones del flujo de eventos no redactados en función del responsable	0 0
Consistencia		
7. ¿El nombre dado a los casos de uso es una expresión verbal que describe alguna funcionalidad relevante en el contexto del usuario?	9. Número de casos de uso que tienen un nombre incorrecto	0
8. ¿Representa el caso de uso una interacción observable por un actor?	10. Número de casos de uso que no representan una interacción observable por un actor	0
9. ¿No existe solapamiento en la funcionalidad que representan los diferentes casos de uso?	11. Número de casos de uso que se solapan	0
10. ¿Existen acciones en el flujo de eventos asignadas a un responsable que no le corresponde?	12. Número de acciones del flujo de eventos que no se corresponde con la definición del responsable 13. Número de casos de uso que tienen acciones del flujo de eventos asignados a un responsable que no le corresponde	0 0
11. ¿Está adecuadamente redactado (en el lenguaje del usuario) el flujo de eventos?	14. Número de casos de uso no aceptados	0
12. Si en el caso de uso interviene más de un actor, ¿existe claridad en cuál de ellos es el actor iniciador?	15. Número de casos de uso con más de un actor que no describe cuál es el actor iniciador	0

13. ¿Existe una adecuada separación entre el flujo básico de eventos y los flujos alternos y/o flujos subordinados?	16. Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos	0
Correctitud		
14. ¿Representa el caso de uso requisitos comprensibles por el usuario?	17. Número de casos de uso en que los requisitos representados no son comprensibles por el usuario	0
15. ¿Se ajusta la representación del diagrama del caso de uso de acuerdo a lo normado en la metodología?	18. Grado en que se ajusta el diagrama del caso de uso a la metodología	0
16. ¿Las interacciones definidas describen la funcionalidad requerida del sistema?	19. Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema	0
Complejidad		
17. ¿En sistemas relativamente grandes se ha realizado una agrupación de los casos de uso en paquetes?	20. Grado en que es adecuada la partición por paquetes	0
18. ¿Los elementos dentro del diagrama están adecuadamente ubicados de manera que facilitan su interpretación?	21. Número de elementos del diagrama que requieren reubicación	0

Anexo.4 : Acta de liberación del módulo Administración del Sistema SINAPSIS.

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.

Martes, 04 de mayo de 2010.

Luego de haber efectuado 2 iteraciones de revisiones a los artefactos: Especificación de Requisitos, Modelo de Sistema y Modelo de diseño del módulo Administración del proyecto Sinapsis del Centro CEGEL de la Facultad 15 y haberse detectado un promedio de 13 No Conformidades, se puede afirmar que se han corregido los defectos encontrados, por lo que se considera que los artefactos están correctamente y listos para ser utilizados.

Firma del Asesor y Jefe del Grupo de Calidad Centro CEGEL

Ing. Raúl Velázquez Alvarez

UCI Universidad de las Ciencias Informáticas
Decanato Facultad

Anexo.5 Acta de Aceptación de la documentación del Módulo de Administración del sistema 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 Administración de Usuario.
- Modelo de Sistema. Módulo de Administración de Usuario.

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

Módulo: Software que agrupa un conjunto de subprogramas y estructuras de datos. Los módulos son unidades que pueden ser compiladas por separado y los hace reusables y permite que múltiples programadores trabajen en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo.

Artefacto: Es un producto de trabajo que proporciona una descripción y una definición para productos de trabajo tangibles

Requisito Funcional: Capacidades o condiciones que el sistema debe cumplir.

Requisitos no Funcional: Propiedades o cualidades que el producto debe tener.

MCUS: Modelo de Caso de Uso del Sistema.

DTD: Definición del Tipo de Documento.

IU: Interfaz de Usuario.

IDE: Entorno de Desarrollo Integrado.

Caso de Uso: Representación de la agrupación de funcionalidades comunes. Representan un conjunto de iteraciones entre el sistema y sus actores.

Persistente: Conjunto de datos y elementos que deber ser almacenados por el espacio de tiempo que se requiera, para dar soporte de información a un sistema u organización.

Framework Hibernate: Herramienta de Mapeo Objeto-Relacional.

Framework Spring: Basado en el principio de inyección de dependencias, hace externo el manejo y la creación de las dependencias de los componentes obteniendo una mayor claridad en el código

PDF: acrónimo del inglés Portable Document Format (Formato de documento portátil), es un formato de almacenamiento de documentos ideado para la impresión de documentos así como su presentación final, no requiriéndose procesos anteriores de ajuste.

Métrica: Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

UML: Acrónimo del inglés Unified Modeling Language (Lenguaje Unificado de Modelado). Es un lenguaje gráfico de modelado de sistemas de software respaldado por el OMG (Object Management Group), se utiliza para visualizar, especificar, construir y documentar un sistema.